cadence™

# Virtuoso Visualization and Analysis XL SKILL Reference

**Product Version ICADVM20.1**
**October 2020**

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

# Contents

Preface . . . . . 11
Scope . . . . . 12
Licensing Requirements . . . . . 12
Related Documentation . . . . . 13
What's New and KPNS . . . . . 13
Installation, Environment, and Infrastructure . . . . . 13
Technology Information . . . . . 14
Virtuoso Tools . . . . . 14
Relative Object Design and Inherited Connections . . . . . 14
Additional Learning Resources . . . . . 15
Video Library . . . . . 15
Virtuoso Videos Book . . . . . 15
Rapid Adoption Kits . . . . . 15
Help and Support Facilities . . . . . 16
Customer Support . . . . . 16
Feedback about Documentation . . . . . 17
Understanding Cadence SKILL . . . . . 18
Using SKILL Code Examples . . . . . 18
Sample SKILL Code . . . . . 18
Accessing API Help . . . . . 19
Typographic and Syntax Conventions . . . . . 19
Identifiers Used to Denote Data Types . . . . . 20

1
Introduction . . . . . 23
Searching for SKILL Functions using Finder Assistant . . . . . 23
Plotting Functions . . . . . 23
Setting Waveform Window Defaults . . . . . 23

# 2
# Waveform Window Functions . . . . . 27


awvAddSubwindow . . . . . 32
awvAnalog2Digital . . . . . 33
awvAppendExpression . . . . . 35
awvAppendList . . . . . 38
awvAppendWaveform . . . . . 41
awvClearPlotWindow . . . . . 45
awvClearSubwindowHistory . . . . . 46
awvClearWindowHistory . . . . . 47
awvCloseCalculator . . . . . 48
awvCloseWindow . . . . . 49
awvCloseWindowMenuCB . . . . . 50
awvCreateBus . . . . . 51
awvCreateBusFromWaveList . . . . . 52
awvCreatePlotWindow . . . . . 53
awvLoadCustomCalcFunction . . . . . 54
awvLoadSharedCustomFunctionsFile . . . . . 57
awvDeleteAllWaveforms . . . . . 58
awvDeleteMarker . . . . . 59
awvDeleteSubwindow . . . . . 60
awvDeleteWaveform . . . . . 61
awvDigital2Analog . . . . . 62
awvDisableRedraw . . . . . 65
awvDisplayDate . . . . . 66
awvDisplayGrid . . . . . 67
awvDisplaySubwindowTitle . . . . . 68
awvDisplayTitle . . . . . 69
awvEraseWindowMenuCB . . . . . 70
awvEval . . . . . 71
awvExitWindowFunctionAdd . . . . . 72
awvExitWindowFunctionDel . . . . . 73
awvExitWindowFunctionGet . . . . . 74
awvGetAssertName . . . . . 75
awvEyeCross . . . . . 76

awvGetCurrentSubwindow . . . . . 79
awvGetCurrentWindow . . . . . 80
awvGetDisplayMode . . . . . 81
awvGetDrawStatus . . . . . 82
awvGetHiWindow . . . . . 83
awvGetInitializationTimeout . . . . . 85
awvGetOnSubwindowList . . . . . 86
awvGetPlotStyle . . . . . 87
awvGetScalarFromWave . . . . . 88
awvGetSelectedTraceWaveforms . . . . . 90
awvGetSmithModeType . . . . . 91
awvGetStripNumberOfSelectedTrace . . . . . 92
awvGetStripNumbersList . . . . . 93
awvGetSubwindowStripCount . . . . . 94
awvGetSubwindowList . . . . . 95
awvGetUnusedEntityList . . . . . 96
awvGetWaveNameList . . . . . 97
awvGetWindowList . . . . . 98
awvGetXAxisLabel . . . . . 99
awvGetXMarkerNames . . . . . 101
awvGetYAxisLabel . . . . . 102
awvGetYMarkerNames . . . . . 104
awvIsnitWindowFunctionAdd . . . . . 105
awvInitWindowFunctionDel . . . . . 107
awvInitWindowFunctionGet . . . . . 108
awvIsPlotWindow . . . . . 109
awvLoadEyeMask . . . . . 110
awvLoadMenuCB . . . . . 111
awvLoadWindow . . . . . 112
awvLogYAxis . . . . . 113
awvLogXAxis . . . . . 115
awvPlaceAMarker . . . . . 116
awvPlaceBMarker . . . . . 118
awvPlaceBookmark . . . . . 120
awvPlaceWaveformLabel . . . . . 124
awvPlaceWindowLabel . . . . . 127

awvPlaceXMarker . . . . . 130
awvPlaceYMarker . . . . . 131
awvPlotExpression . . . . . 133
awvPrintWaveform . . . . . 137
awvPlotList . . . . . 141
awvPlotSignals . . . . . 147
awvPlotSimpleExpression . . . . . 149
awvPlotWaveform . . . . . 151
awvPlotWaveformOption . . . . . 157
awvRedisplaySubwindow . . . . . 160
awvRedrawWindowMenuCB . . . . . 161
awvRedisplayWindow . . . . . 162
awvRemoveDate . . . . . 163
awvResumeViVA . . . . . 164
awvRemoveLabel . . . . . 165
awvRemoveSubwindowTitle . . . . . 166
awvRemoveTitle . . . . . 167
awvResetAllWindows . . . . . 168
awvResetWindow . . . . . 169
awvRfLoadPull . . . . . 171
awvSaveWindow . . . . . 173
awvSaveWindowImage . . . . . 174
awvSaveMenuCB . . . . . 175
awvSaveToCSV . . . . . 176
awvSetCurrentSubwindow . . . . . 179
awvSetCurrentWindow . . . . . 180
awvSetCursorPrompts . . . . . 181
awvSetDisplayMode . . . . . 182
awvSetDisplayStatus . . . . . 183
awvSetInitializationTimeout . . . . . 184
awvSetLegendWidth . . . . . 185
awvSetOptionDefault . . . . . 186
awvSetOptionValue . . . . . 187
awvSetOrigin . . . . . 188
awvSetPlotStyle . . . . . 189
awvSetSmithModeType . . . . . 190

awvSetSmithXLimit . . . . . 192
awvSetSmithYLimit . . . . . 194
awvSmithAxisMenuCB . . . . . 196
awvSetUpdateStatus . . . . . 197
awvSetWaveformDisplayStatus . . . . . 198
awvSetWaveNameList . . . . . 199
awvSetXAxisLabel . . . . . 200
awvSetXLimit . . . . . 201
awvSetXScale . . . . . 202
awvSetYAxisLabel . . . . . 203
awvSetYLimit . . . . . 205
awvSetYRange . . . . . 207
awvSimplePlotExpression . . . . . 209
awvTableSignals . . . . . 212
awvUpdateAllWindows . . . . . 213
awvUpdateWindow . . . . . 214
awvZoomFit . . . . . 215
awvZoomGraphX . . . . . 216
awvZoomGraphY . . . . . 217
awvZoomGraphXY . . . . . 219
awvGetSubwindowTitle . . . . . 221
awvGetWindowTitle . . . . . 222
awvGetXAxisMajorDivisions . . . . . 223
awvGetXAxisMinorDivisions . . . . . 224
awvGetXAxisStepValue . . . . . 225
awvGetXAxisUseStepValue . . . . . 226
awvSetXAxisMajorDivisions . . . . . 227
awvSetXAxisMinorDivisions . . . . . 228
awvSetXAxisStepValue . . . . . 229
awvSetXAxisUseStepValue . . . . . 230
awvGetYAxisMajorDivisions . . . . . 232
awvGetYAxisMinorDivisions . . . . . 233
awvGetYAxisStepValue . . . . . 234
awvGetYAxisUseStepValue . . . . . 235
awvSetYAxisMajorDivisions . . . . . 236
awvSetYAxisMinorDivisions . . . . . 237

awvSetYAxisStepValue . . . . . 238
awvSetYAxisUseStepValue . . . . . 239
OT . . . . . 240
OS . . . . . 241
pvrfreq . . . . . 242
pvifreq . . . . . 244
firstVal . . . . . 247
lastVal . . . . . 248
valueAt . . . . . 249
eyeMask . . . . . 251
eyeMaskViolationPeriodCount . . . . . 253
eyeBERLeft . . . . . 254
eyeBERRight . . . . . 255
eyeBERLeftApprox . . . . . 256
eyeBERRightApprox . . . . . 257

3
Results Browser Functions . . . . . 259

rdbLoadResults . . . . . 260
rdbReloadResults . . . . . 261
rdbUnloadResults . . . . . 262
rdbSetCurrentDirectory . . . . . 263
rdbWriteToFormat . . . . . 264
rdbShowDialog . . . . . 265
vvDisplayBrowser . . . . . 266
vivaInitBindkeys . . . . . 267

4
Callback Functions . . . . . 269

awviEditMenuCB . . . . . 270
awviMakeActiveMenuCB . . . . . 271
awviPLoadMenuCB . . . . . 272
awviPSaveMenuCB . . . . . 273
awviPUpdateMenuCB . . . . . 274
awviShowOutputMenuCB . . . . . 275

appendWaves . . . . . 276
waveVsWave . . . . . 277

5
Calculator Functions . . . . . 279

armSetCalc . . . . . 280
calCalculatorFormCB . . . . . 281
calCalcInput . . . . . 282
calCreateSpecialFunction . . . . . 284
calCreateSpecialFunctionsForm . . . . . 285
calGetBuffer . . . . . 286
calSetBuffer . . . . . 287
calSetCurrentTest . . . . . 288
caliModeToggle . . . . . 289
caliRestoreDefaultWindowSize . . . . . 290
calRegisterSpecialFunction . . . . . 291
calSpecialFunctionInput . . . . . 292
expr . . . . . 294
famEval . . . . . 295
vvDisplayCalculator . . . . . 296
adtFFT . . . . . 297
adtIFFT . . . . . 298
topLine . . . . . 299
baseLine . . . . . 300
topBaseLine . . . . . 301
leafValue . . . . . 302
swapSweep . . . . . 304
rfEdgePhaseNoise . . . . . 305
rfInputNoise . . . . . 307
rfOutputNoise . . . . . 308
rfTransferFunction . . . . . 309
numConv . . . . . 310
busTransition . . . . . 311
aaSP . . . . . 314
rfJitter . . . . . 315

rfJc . . . . . 317
rfJcc . . . . . 319
rfThresholdXing . . . . . 321
rfWrlsCim3Value . . . . . 322
rfWrlsCim5Value . . . . . 323
rfWrlsMeasContour . . . . . 324
rfWrlsCcdfValues . . . . . 326
rfCimMcpValue . . . . . 327
rfGetMinDampFactor . . . . . 328
rfGetEventtimeIndex . . . . . 329
eyeHeightAtXY . . . . . 331
eyeWidthAtXY . . . . . 332
triggeredDelay . . . . . 333
mu . . . . . 336
Mu . . . . . 338
mu_prime . . . . . 340
Mu_prime . . . . . 342

# Preface

The Virtuoso® Visualization and Analysis XL is an analog and mixed-signal waveform display tool. This user guide describes this tool in detail and explains how you can use the various features of this tool.

The tool helps you analyze the data generated by your simulator.

The tool consists of the following components:

- Results Browser displays simulation data in the hierarchical arrangement of your design.
- Graph offers features that simplify the processing of your signal data.
- Calculator provides an extensive expression building capability that addresses the needs of a wide variety of analysis types.

This user guide is aimed at developers and designers of integrated circuits and assumes that you are familiar with:

- The Virtuoso design environment and application infrastructure mechanisms designed to support consistent operations between all Cadence® tools.
- The applications used to design and develop integrated circuits in the Virtuoso design environment, notably, the Virtuoso Layout Suite, and Virtuoso Schematic Editor.
- The Virtuoso design environment technology file.
- Component description format (CDF), which lets you create and describe your own components for use with Layout XL

The preface contains the following topics:


- Scope
- Licensing Requirements
- Related Documentation
- Additional Learning Resources
- Customer Support
- Feedback about Documentation

- Understanding Cadence SKILL
- Typographic and Syntax Conventions
- Identifiers Used to Denote Data Types

# Scope

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

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

# Licensing Requirements

Following is the licensing scheme for the Virtuoso Visualization and Analysis XL tool:

***Virtuoso Visualization and Analysis XL when opened from ADE Explorer, Assembler, VVOI, L, XL, and GXL:***

- Shares license tokens with ADE L, XL, or GXL.
- When you close the ADE window, the Virtuoso Visualization and Analysis XL continues to hold the ADE license tokens, which are in effect until all Virtuoso Visualization and Analysis XL windows are closed.

***Virtuoso Visualization and Analysis XL when opened in stand-alone mode or from Virtuoso:***

- Checks out either the Virtuoso Visualization and Analysis XL license or an ADE license tier, depending on the preferences you have set by using the `VIVA License Checkout Order` .cdsenv variable. By default, this variable is set to `ViVA, ADE`, which results in the following license check out tasks being performed:

- Checks out the Virtuoso Visualization and Analysis XL license, if available.
- If the check out operation in the previous step fails, you can choose between checking out an ADE license tier or two ADE GXL tokens, based on the order set in the `ADELicenseCheckoutOrder` .cdsenv variable, which controls the order in which ADE license tiers are checked out.

If the `VIVALicenseCheckoutOrder` variable is set to `ADE, ViVA`, the license check out tasks are performed in the following order:

- Checks out an ADE license tier or two ADE GXL tokens, based on the order set in the `ADELicenseCheckoutOrder` .cdsenv variable.
- If the check out operation in the previous step fails, you can check out the Virtuoso Visualization and Analysis XL license.

- The license is released when all the Virtuoso Visualization and Analysis XL windows are closed.

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

# Related Documentation

The following documents provide more information about the topics discussed in this guide.

## What's New and KPNS

- *Virtuoso Visualization and Analysis XL What's New*
- *Virtuoso Visualization and Analysis XL KPNS*

## Installation, Environment, and Infrastructure

- *Cadence Installation Guide*
- *Virtuoso Design Environment User Guide*
- *Virtuoso Design Environment SKILL Reference*
- *Cadence Application Infrastructure User Guide*

## Technology Information

- *Virtuoso Technology Data User Guide*
- *Virtuoso Technology Data ASCII Files Reference*
- *Virtuoso Technology Data SKILL Reference*

## Virtuoso Tools

- *Virtuoso ADE Explorer and Assembler SKILL Reference*
- *Virtuoso ADE Assembler User Guide*
- *Virtuoso Analog Design Environment XL User Guide*
- *Virtuoso Analog Design Environment GXL User Guide*
- *Virtuoso Visualization and Analysis XL User Guide*
- *Virtuoso Analog Distributed Processing Option User Guide*
- *Virtuoso Parasitic Estimation and Analysis User Guide*
- *Virtuoso Schematic Editor L User Guide*
- *Spectre Circuit Simulator Reference*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator RF Analysis User Guide*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator User Guide*
- *Virtuoso UltraSim Simulator User Guide*
- *Component Description Format User Guide*
- *Analog Expression Language Reference*
- *Virtuoso Design Environment User Guide*
- *Virtuoso Design Environment SKILL Reference*
- *Virtuoso® Spectre® Circuit Simulator Reference.*

## Relative Object Design and Inherited Connections

- *Virtuoso Relative Object Design User Guide*

- *Virtuoso Schematic Editor L User Guide*

# Additional Learning Resources

## Video Library

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

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

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

## Virtuoso Videos Book

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

## Rapid Adoption Kits

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

In addition, Cadence offers the following training courses on Virtuoso Visualization and Analysis XL:

- Virtuoso Analog Design Environment
- Virtuoso Schematic Editor
- Analog Modeling with Verilog-A
- Behavioral Modeling with Verilog-AMS
- Real Modeling with Verilog-AMS
- Spectre Simulations Using Virtuoso ADE

- Virtuoso UltraSim Full-Chip Simulator
- Virtuoso Simulation for Advanced Nodes

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

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

### Help and Support Facilities

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

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

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

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

## Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

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

- Log on to Cadence Online Support

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

# Feedback about Documentation

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

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

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

# Understanding Cadence SKILL

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

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

## Using SKILL Code Examples

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

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

## Sample SKILL Code

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

**axlGetRunStatus**

```
axlGetRunStatus(
    t_sessionName                          <- Required argument
    [ ?optionName t_optionName ]           <- Optional keyword argument
    [ ?historyName t_historyName ]         <- Optional keyword argument
    )
    => l_statusValues                      <- Return value
```

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

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

**Example**

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

Return value

Value of the session name argument

Question mark and argument name before the value of the keyword argument

Question mark and argument name before the value of the keyword argument

## Accessing API Help

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

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

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

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

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

# Typographic and Syntax Conventions

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

| | |
|---|---|
| *text* | Indicates names of manuals, menu commands, buttons, and fields. |
| `text` | Indicates text that you must type as presented. Typically used to denote command, function, routine, or argument names that must be typed literally. |

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

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

# Identifiers Used to Denote Data Types

Data type identifiers are used to indicate the type of value required by an API argument. These data types are denoted by a single letter that is prefixed to the argument label and is separated from the argument by an underscore; for example, *t* is the data type in

`t_viewName`. Data types and underscores are used only as identifiers; they must not be typed when specifying the argument in a function.

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

| Prefix | Internal Name | Data Type |
|---|---|---|
| *t* | string | character string (text) |
| *T* | txobject | transient object |
| *u* | function | function object, either the name of a function (symbol) or a lambda function body (list) |
| *U* | funobj | function object |
| *v* | hdbpath | hdbpath |
| *w* | wtype | window type |
| *sw* | swtype | subtype session window |
| *dw* | dwtype | subtype dockable window |
| *x* | integer | integer number |
| *y* | binary | binary function |
| *&* | pointer | pointer type |

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

# 1

# Introduction

The Virtuoso Visualization and Analysis XL SKILL functions in this manual are organized into sections according to their purpose. Within each chapter, the functions are in alphabetical order.

Each Waveform Window can contain a number of subwindows, each identified by an integer index. Each subwindow can contain a number of curves, each identified by an integer index.

Many of the functions in this chapter describe operations you can perform on a subwindow. If you do not specify a particular subwindow, the current subwindow is affected. It is important to understand that even if you call a Waveform Window and do not add any subwindows, your waveform display is still considered a subwindow. You can think of it as subwindow number one of one. Any of the functions that deal with subwindows will work on this type of waveform display as well as on waveform displays with several subwindows.

## Searching for SKILL Functions using Finder Assistant

To view the description and syntax of the Virtuoso Visualization and Analysis XL SKILL functions, use the *Finder* assistant that can be accessed either directly from the CIW or through the SKILL IDE tool. For information about using the Finder assistant, refer to the Working with the Finder Assistant section in the *Cadence SKILL IDE User Guide*.

## Plotting Functions

By default, waveforms are plotted in the current subwindow, go to axis Y1, and to different strips (when the display mode is `strip`).

## Setting Waveform Window Defaults

You can set the default values for some of the Waveform Window options with the `awvSetOptionValue` function. When you set the default of an option in the CIW, the new value takes effect when you open a new Waveform Window or add a subwindow to an existing

Waveform Window. The plotting options take effect as soon as you send an image to the plotter.

If you want the defaults to apply to every new session, you need to change the options in your `.cdsinit` file.

| Option name | Description | Type | Default | Valid values |
|---|---|---|---|---|
| cursorSuppressed | Removes the tracking cursor display | Boolean | `nil` | `t or nil` |
| cursorPrecision | Controls the number of digits displayed (at the top of the window) as cursor output | integer | `4` | Any integer greater than 2 and less than 16 |
| cursorAction | Controls whether the cursor snaps to waveforms or original data points | string | `"line"` | `"data point"` `"line"` |
| cursorPhase | Controls the phase display (Smith) | string | `"degree"` | `"degree"` `"radian"` |
| cursorValue | Controls the value display (Smith) | string | `"normalized impedance"` | `"normalized admittance"` `"reflection coefficient"` `"normalized impedance"` |
| dateStamp | Adds the date to the top right corner of the window | Boolean | `nil` | `t or nil` |
| displayAxes | Displays the axes | Boolean | `t` | `t or nil` |

| Option name | Description | Type | Default | Valid values |
|---|---|---|---|---|
| displayAxesBy125 | Displays the axis labels by increments of 1, 2, or 5 | Boolean | `nil` | `t or nil` |
| displayAxesLabel | Displays the axes labels | Boolean | `t` | `t or nil` |
| displayGrids | Displays grid lines | Boolean | `nil` | `t or nil` |
| displayMajorTicks | Displays major tick marks | Boolean | `t` | `t or nil` |
| displayMinorTicks | Displays minor tick marks | Boolean | `t` | `t or nil` |
| xLog | Displays the X axis logarith-mically | Boolean | `nil` | `t or nil` |
| mode | Controls the mode type | string | `"composite"` | `"strip" "smith" "composite"` |
| style | Controls how waveforms plot in the window | string | `"auto"` | `"bar" "scatterPlot""joined" "auto"` |
| numIdentifier | Controls the number of identifiers per waveform | integer | `6` | Any positive integer or nil to show all the identifiers |
| hcCopyNum | Specifies the number of copies to plot | integer | `1` | Any positive integer |
| hcDisplay | Specifies the display name | string | `"display"` | Defined in the technology file |
| hcHeader | Specifies header with plot | boolean | t | t or nil |

| Option name | Description | Type | Default | Valid values |
|---|---|---|---|---|
| hcMailLogNames | Emails plot submission output to user | boolean | t | t or nil |
| hcOrientation | Specifies the plot orientation | string | `"automatic"` | `"portrait"`<br>`"landscape"`<br>`"automatic"` |
| hcOutputFile | Specifies to plot to the file only | general (string or nil) | `nil` | Name of the output file |
| hcPaperSize | Specifies the plot paper size | string | Specified in `.cdsplotinit` | |
| hcPlotterName | Sets the plotter name | string | Specified in `.cdsplotinit` | |
| hcTmpDir | Specifies the temporary scratch space | string | `"/usr/tmp"` | Name of a temporary directory |

# 2

# Waveform Window Functions

This chapter describes the following waveform window functions in detail:

- awvAddSubwindow
- awvAnalog2Digital
- awvAppendExpression
- awvAppendList
- awvAppendWaveform
- awvClearPlotWindow
- awvClearSubwindowHistory
- awvClearWindowHistory
- awvCloseCalculator
- awvCloseWindowMenuCB
- awvCreateBus
- awvCreateBusFromWaveList
- awvCreatePlotWindow
- awvDeleteAllWaveforms
- awvDeleteMarker
- awvDeleteSubwindow
- awvDeleteWaveform
- awvDigital2Analog
- awvDisableRedraw
- awvDisplayDate

- awvDisplayGrid
- awvDisplaySubwindowTitle
- awvDisplayTitle
- awvEraseWindowMenuCB
- awvEval
- awvExitWindowFunctionAdd
- awvExitWindowFunctionDel
- awvExitWindowFunctionGet
- awvGetCurrentSubwindow
- awvGetCurrentWindow
- awvGetDisplayMode
- awvGetDrawStatus
- awvGetInitializationTimeout
- awvGetOnSubwindowList
- awvGetPlotStyle
- awvGetScalarFromWave
- awvGetStripNumberOfSelectedTrace
- awvGetStripNumbersList
- awvGetSubwindowStripCount
- awvGetSelectedTraceWaveforms
- awvGetSmithModeType
- awvGetSubwindowList
- awvGetUnusedEntityList
- awvGetWaveNameList
- awvGetWindowList
- awvGetXAxisLabel
- awvGetXMarkerNames

- awvGetYAxisLabel
- awvGetYMarkerNames
- awvIsnitWindowFunctionAdd
- awvInitWindowFunctionDel
- awvInitWindowFunctionGet
- awvIsPlotWindow
- awvLoadMenuCB
- awvLoadWindow
- awvLogYAxis
- awvLogXAxis
- awvPlaceBookmark
- awvPlaceWaveformLabel
- awvPlaceWindowLabel
- awvPlaceXMarker
- awvPlaceYMarker
- awvPlotExpression
- awvPrintWaveform
- awvPlotList
- awvPlotWaveform
- awvPlotWaveformOption
- awvRedisplaySubwindow
- awvRedrawWindowMenuCB
- awvRedisplayWindow
- awvRemoveDate
- awvResumeViVA
- awvRemoveLabel
- awvRemoveSubwindowTitle

- awvRemoveTitle
- awvResetAllWindows
- awvResetWindow
- awvRfLoadPull
- awvSaveWindow
- awvSaveWindowImage
- awvSaveMenuCB
- awvSaveToCSV
- awvSetCurrentSubwindow
- awvSetCurrentWindow
- awvSetCursorPrompts
- awvSetDisplayMode
- awvSetDisplayStatus
- awvSetInitializationTimeout
- awvSetOptionDefault
- awvSetOptionValue
- awvSetOrigin
- awvSetPlotStyle
- awvSetSmithModeType
- awvSetSmithModeType
- awvSetSmithXLimit
- awvSetSmithYLimit
- awvSmithAxisMenuCB
- awvSetUpdateStatus
- awvSetWaveformDisplayStatus
- awvSetWaveNameList
- awvSetXAxisLabel

- awvSetXLimit
- awvSetXScale
- awvSetYAxisLabel
- awvSetYLimit
- awvSetYRange
- awvSimplePlotExpression
- awvUpdateAllWindows
- awvUpdateWindow
- awvZoomFit
- awvZoomGraphX
- awvZoomGraphY
- awvZoomGraphXY

*Caution*

***When you work with the Waveform Window using SKILL, you must use* only *the functions described in this chapter. Never use the functions that appear in the CIW when you use the menus in the Waveform Window because these functions interact with menus and forms directly. Unpredictable results might occur if you use these CIW functions in a SKILL procedure and their associated forms and menus are not instantiated.***

## awvAddSubwindow

```
awvAddSubwindow(
     w_windowId
     )
     => x_subwindow | nil
```

### Description

Adds a subwindow to the Waveform window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |

### Value Returned

| | |
|---|---|
| *x_subwindow* | Returns the number for the new subwindow (found in the upper right corner). |
| `nil` | Returns `nil` if the specified Waveform window does not exist. |

### Example

```
awvAddSubwindow( window(3) )
=> 2
```

Adds a new subwindow and returns its number.

### Related Function

To delete a subwindow, see the awvDeleteSubwindow function on page-60.

## awvAnalog2Digital

```
awvAnalog2Digital(
     o_wave
     n_vhi
     n_vlo
     n_vc
     n_timex
     t_thresholdType
     )
     => o_digWave / n_digval / nil
```

### Description

Returns the digital form of the analog input, which can be a waveform, list or family of waveforms, or a string representation of expression(s).

### Arguments

| | |
|---|---|
| *o_wave* | Input waveform. |
| *o_vhi* | High threshold value (used only when *t_thresholdType* is `hilo`). |
| *o_vlo* | Low threshold value (used only when *t_thresholdType* is `hilo`). |
| *o_vc* | Central threshold value (used only when *t_thresholdType* is `centre`). |
| *n_timex* | The value that determines logic X. |
| *t_thresholdType* | Can take the values `hilo` or `centre`. If *t_thresholdType* is `centre`, it is a high state (1) unless its value is less than `n_vc`, in which case it is a low state (0). If *t_thresholdType* is hilo, any value less than *n_vlo* is a low state (0), any value greater than *n_vhi* is a high state (1) and the rest is treated as unknown based on the value of *n_timex*. |

### Value Returned

| | |
|---|---|
| *o_digWave* | A waveform (or a list of waveforms) is returned if the analog input specified was *o_wave*. |

| | |
|---|---|
| *o_digVal* | A scalar value is returned if the analog input specified was *o_val*. |
| nil | Returns nil if the specified Waveform window does not exist. |

## awvAppendExpression

```
awvAppendExpression(
     w_windowId
     t_expr
     l_context
     [ ?index       l_waveIndexList ]
     [ ?color       l_colorList ]
     [ ?lineType    l_styleList ]
     [ ?dataSymbol  l_symbolList ]
     [ ?subwindow   x_subwindow ]
     [ ?showSymbols l_showList ]
     [ ?lineStyle   l_styleList ]
     [ ?lineThickness l_thicknessList ]
     )
     => t / nil
```

### Description

Evaluates the *t_expr* expression and adds the resulting waveforms to a subwindow.

The new waveforms are plotted at the same strip and Y axis as their corresponding curve from *l_waveIndexList*. Also, the new waveforms are assigned the next lowest unassigned numbers. So, if *l_waveIndexList* contains curves 1 and 2, the curves resulting from the expression evaluation are numbered 3, 4, and so on.

**Note:** If you do not specify *l_waveIndexList*, the new waveforms are plotted at the Y-axis and strip of existing waveforms with the lowest numbers. These new waveforms are assigned the lowest *unused* numbers.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *t_expr* | String containing an expression that is evaluated once when the command is issued and re-evaluated at the completion of each simulation in the auto-update mode. *t_expr* can be `nil`. |
| *l_context* | Data context for a particular simulation. If evaluating the expressions requires data generated during a simulation, you must specify this argument. Otherwise, specify `nil`. |
| `?index` *l_waveIndexList* | List of integer identifiers for existing waveform curves that were plotted with one of the Waveform Window plot functions |
| `?color` *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.<br>Valid Values: "`y1`" through "`y66`". |
| `?lineType` *l_styleList* | List specifying the line styles for the waveforms. If you do not supply this argument, the default line style is used.<br>Valid Values: `line`, `bar`, `scalarPlot` and `polezero`.<br>Default Value: `Line`. |
| `?dataSymbol` *l_symbolList* | List of symbols to use if you want the system to plot data points only (that is, not curves). The system places a symbol on each data point and does not use a line to connect the points. To use a symbol, specify an integer or a single character corresponding to the symbol, as listed in Symbol List. |
| *?subwindow* *x_subwindow* | Number for the subwindow (found in the upper right corner).<br>Default Value: Current subwindow |
| `?showSymbol` *l_showList* | Flag or list of flags that specifies if the symbols for the traces should be plotted or not. By default, the value is set as `nil` and none of the symbols is plotted. Set the value as `t` to show symbols for all the traces. To set the show status for each symbol, specify a list of flags, one flag for each symbol. The number of flags should match the number of symbols in `l_symbolList`. Each flag specifies if the corresponding symbol in the `l_symbolList` list will be shown or not.<br>`Default Value: nil` |
| `?LineStyle` *l_styleList* | Specifies the line-style of the signal.Valid values: `Solid`, `Dotted`, `Dashed`, `Dotdashed` |

| | |
|---|---|
| `?lineThickness` *`l_thicknessList`* | Specifies the thickness of the signal. Valid values: `Fine`, `Medium`, `Bold` |

Symbol List

| To use the symbol | Enter integers | Enter character |
|---|---|---|
| + (plus) | 0, 10 or 20 | + |
| . (dot) | 1 | . |
| x | 2, 9, 19 | x or X |
| square | 3, 5, 15 | Not supported |
| circle | 4, 6 or 16 | o or O |
| box | 5 | Not supported |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the *`t_expr`* expression is evaluated and the resulting waveforms are added to the Waveform Window. |
| `nil` | Returns `nil` if there is an error. |

### Related Functions

To plot the Y values in a list against the X values in a list and add the resulting waveforms to a subwindow, see awvAppendList.

To add the waveforms in a list to a subwindow, see awvAppendWaveform.

To evaluate an expression and assign the numbers specified in a list to the waveforms resulting from the evaluation, see awvPlotExpression.

To plot the Y values in a list against the X values in a list and display the resulting waveforms in a subwindow, see awvPlotList.To plot the waveforms in a list in a subwindow, see awvPlotWaveform.

To evaluate an expression and specify whether to add the resulting waveforms to the set of existing waveforms, or overwrite the existing waveforms, see awvSimplePlotExpression.

## awvAppendList

```
awvAppendList(
     w_windowId
     l_YListList
     l_XList
     [ ?index       l_waveIndexList ]
     [ ?color       l_colorList ]
     [ ?lineType    l_styleList ]
     [ ?dataSymbol  l_symbolList ]
     [ ?subwindow   x_subwindow ]
     )
     => t | nil
```

### Description

Plots the Y values in *l_YListList* against the X values in *l_XList* and adds the resulting waveforms to a subwindow.

The new waveforms are plotted at the same strip and Y axis as their corresponding curve in *l_waveIndex*. Also, the new waveforms are assigned the next lowest unassigned numbers. So, if *l_waveIndexList* contains curves 1 and 2, the curves resulting from plotting the X and Y values are numbered 3, 4, and so on.

If you do not specify *l_waveIndexList*, the new waveforms are plotted at the Y axis and strip of existing waveforms with the lowest numbers. These new waveforms are assigned the lowest *unused* numbers.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *l_YListList* | List of lists that specify the different Y values. Each list must have the same number of elements. |
| *l_XList* | List that specifies the different X values. The number of elements in this list must equal the number of elements in each list in *l_YListList* |
| ?index *l_waveIndexList* | List of integers identifying existing waveform curves that were plotted with one of the Waveform window plot functions |
| ?color *l_colorList* | List specifying the colors for the waveforms. If you do not supply this argument, the last color used in each of the waveform entities is used. The colors that are available are defined in your technology file.<br>Valid Values: "y1" through "y66" |
| ?lineType *l_styleList* | List specifying the line styles for the waveforms. If you do not supply this argument, the default line style is used.<br>Valid Values: line, bar, scalarPlot and polezero. Default Value: Line. |
| ?dataSymbol *l_symbolList* | List of symbols to use if you want the system to plot data points only (that is, not curves). The system places a symbol on each data point and does not use a line to connect the points. To use a symbol, specify an integer or a single character corresponding to the symbol, as listed in Symbol List. |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow. |

Symbol List

| To use the symbol | Enter integers | Enter character |
|---|---|---|
| + (plus) | 0, 10 or 20 | + |
| . (dot) | 1 | . |
| x | 2, 9, 19 | x or X |
| square | 3, 5, 15 | Not supported |
| circle | 4, 6 or 16 | o or O |
| box | 5 | Not supported |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the waveforms created by plotting the specified Y values against the specified X values are added to the Waveform window. |
| `nil` | Returns `nil` if there is an error. |

### Related Functions

To evaluate an expression and add the resulting waveforms to a subwindow, see awvAppendExpression.

To add the waveforms in a list to a subwindow, see awvAppendWaveform.

To evaluate an expression and assign the numbers specified in a list to the waveforms resulting from the evaluation, see awvPlotExpression.

To plot the Y values in a list against the X values in a list and display the resulting waveforms in a subwindow, see awvPlotList.

To plot the waveforms in a list in a subwindow, see awvPlotWaveform.

To evaluate an expression and specify whether to add the resulting waveforms to the set of existing waveforms, or overwrite the existing waveforms, see awvSimplePlotExpression.

## awvAppendWaveform

```
awvAppendWaveform(
     w_windowId
     l_waveform
     [ ?expr        l_exprList ]
     [ ?index       l_waveIndexList ]
     [ ?color       l_colorList ]
     [ ?lineType    l_styleList ]
     [ ?dataSymbol  l_symbolList ]
     [ ?subwindow   x_subwindow ]
     [ ?stripNumber l_stripNumberList ]
     [ ?showSymbols l_showList ]
     [ ?lineStyle   l_styleList ]
     [ ?lineThickness l_thicknessList ]
     )
     => t / nil
```

### Description

Adds the waveforms in the *l_waveform* list to a subwindow.

The new waveforms are plotted at the same strip and Y axis as their corresponding curves from *l_waveIndexList*. Also, the new waveforms are assigned the next lowest unassigned numbers. So, if *l_waveIndexList* contains curves 1 and 2, the new curves are numbered 3, 4, and so on.

If you do not specify *l_waveIndexList*, the new waveforms are plotted at the Y axis and strip of existing waveforms with the lowest numbers. These new waveforms are assigned the lowest *unused* numbers.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *l_waveform* | List of waveforms to add. |
| *?expr l_exprList* | Specifies the expressions to display next to the waveform identifiers. If you do not specify *l_exprList*, no expressions are displayed beside the waveform identifiers. |
| ?index *l_waveIndexList* | List of integer identifiers for existing waveform curves that were plotted with one of the Waveform window plot functions. |
| ?color *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" |
| ?lineType *l_styleList* | List specifying the line styles for the waveforms. If you do not supply this argument, the default line style is used. Valid Values: line, bar, scalarPlot and polezero. Default Value: Line. |
| ?dataSymbol *l_symbolList* | List of symbols to use if you want the system to plot data points only (that is, not curves). The system places a symbol on each data point and does not use a line to connect the points. To use a symbol, specify an integer or a single character corresponding to the symbol, as listed in Symbol List. |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |
| ?stripNumber *l_stripNumberList* | List of numbers identifying the strips. |
| ?showSymbols *l_showList* | Flag or list of flags that specifies if the symbols for the traces should be plotted or not. By default, the value is set as nil and none of the symbols is plotted. Set the value as t to show symbols for all the traces. To set the show status for each symbol, specify a list of flags, one flag for each symbol. The number of flags should match the number of symbols in l_symbolList. Each flag specifies if the corresponding symbol in the l_symbolList list will be shown or not. Default Value: nil |
| ?lineStyle *l_styleList* | Specifies the line-style of the signal. Valid values: Solid, Dotted, Dashed, Dotdashed |
| ?lineThickness *l_thicknessList* | Specifies the thickness of the signal. Valid values: Fine, Medium, Bold |

Symbol List

| To use the symbol | Enter integers | Enter character |
|---|---|---|
| + (plus) | 0, 10 or 20 | + |
| . (dot) | 1 | . |
| x | 2, 9, 19 | x or X |
| square | 3, 5, 15 | Not supported |
| circle | 4, 6 or 16 | o or O |
| box | 5 | Not supported |

### Value Returned

| | |
|---|---|
| `t` | Returns t when the *`l_waveform`* waveforms are added to the Waveform window. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvAppendWaveform( window( 2 ) list( w3 w4 ) ?color
    list( "y2" "y4" )
)
```

Adds waveforms *w3* and *w4* to the strip and Y axis locations of waveforms 1 and 2, respectively. *w3* is displayed in the *y2* layer color, and *w4* is displayed in the *y4* layer color.

### Related Functions

To evaluate an expression and add the resulting waveforms to a subwindow, see the awvAppendExpression function on page-35.

To plot the Y values in a list against the X values in a list, see the awvAppendList function on page-38.

To evaluate an expression and assign the numbers specified in a list to the waveforms resulting from the evaluation, see the awvPlotExpression function on page-133.

To plot the Y values in a list against the X values in a list and display the resulting waveforms in a subwindow, see the awvPlotList function on page-141.

To plot the waveforms in a list in a subwindow, see the awvPlotWaveform function on page-151.

To evaluate an expression and specify whether to add the resulting waveforms to the set of existing waveforms, or overwrite the existing waveforms, see the awvSimplePlotExpression function on page-209.

## awvClearPlotWindow

```
awvClearPlotWindow(
     w_windowId
     )
     => t / nil
```

### Description

Clears the graphics shown in the waveform window. The History for the window and subwindows are maintained.

### Arguments

| | |
|---|---|
| *w_windowId* | window ID. |

### Value Returned

| | |
|---|---|
| *t* | Returns `t` when the contents of the window are erased. |
| *nil* | Returns `nil` if the Waveform window does not exist. |

### Example

```
awvClearPlotWindow(awvGetCurrentWindow()) => t
```

## awvClearSubwindowHistory

```
awvClearSubwindowHistory(
     w_windowId
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Erases the contents of a particular subwindow. This function deletes the waveforms, title, date stamp, and labels stored in internal memory. The other subwindows are not affected.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?subwindow<br>*x_subwindow* | Number for the subwindow (found in the upper right corner).<br>Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| t | Returns t when the contents of the subwindow are erased. |
| nil | Returns nil if the Waveform window or the subwindow does not exist. |

### Example

```
awvClearSubwindowHistory( window(3) ?subwindow 1 )
=> t
```

Erases the contents of subwindow 1 and deletes the information from internal memory.

### Related Function

To erase the contents of a Waveform window, see the awvClearWindowHistory function on page-47.

## awvClearWindowHistory

```
awvClearWindowHistory(
     w_windowId
     [ ?force  g_force ]
     )
     => t / nil
```

### Description

Erases the contents of a Waveform window and deletes the waveforms, title, date stamp, and labels stored in internal memory. This function operates on subwindows whose update statuses are *on*. To force this function to operate on all subwindows, set `g_force` to `t`.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?force *g_force* | Boolean flag that specifies whether the function operates on all subwindows, or only on subwindows whose update statuses are turned *on*.<br>Valid Values: `t` to perform function on *all* subwindows, or `nil` to perform function only on updatable subwindows.<br>Default value: `nil`. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the waveform information is deleted. |
| `nil` | Returns `nil` if the specified Waveform window does not exist. |

### Example

```
awvClearWindowHistory( window(3) ?force t )
=> t
```

Clears all the subwindows from Waveform window 3, including subwindows whose update statuses are turned *off*.

### Related Function

To erase the contents of a subwindow, see the awvClearSubwindowHistory.

## awvCloseCalculator

```
awvCloseCalculator(
     adesession
     adexlSession
     )
     => t / nil
```

### Description

Closes calculator window of the current session or the session specified (optional).

### Arguments

| | |
|---|---|
| *adeSession* | Optional; closes the calculator window invoked from ADE session. |
| *adexlSession* | Optional; closes the calculator window invoked from ADEXL session |

### Value Returned

| | |
|---|---|
| *t* | Returns `t` if the command was successful. |
| *nil* | Otherwise, returns nil. |

### Example

```
awvCloseCalculator()
```

## awvCloseWindow

```
awvCloseWindow(
     w_windowID
     )
     => t / nil
```

### Description

Closes the specified Waveform window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |

### Value Returned

| | |
|---|---|
| *t* | Returns `t` if the command was successful. |
| *nil* | Otherwise, returns nil. |

### Example

```
awvCloseWindow(window(6))
```

## awvCloseWindowMenuCB

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

### Description

Closes the current window.

The function is defined in `dfII/etc/context/awv.cxt`.

### Arguments

None

### Value Returned

| | |
|---|---|
| *t* | Returns `t` if the command was successful. Otherwise, throws error (if there are no windows to be deleted). |

### Example

```
awvCloseWindowMenuCB() => t
```

## awvCreateBus

```
awvCreateBus(
     w_bus
     l_wavelist
     r_radix
     )
     => o_bus / nil
```

### Description

Creates a bus with the given digital signals and radix type.

### Arguments

| | |
|---|---|
| *w_bus* | Name of the digital waveform representing a bus. |
| *l_wavelist* | List of digital waves or expressions in the bus |
| *r_radix* | Radix of the bus.<br>Valid values : `Binary`, `Octal`, `Ascii`, `Hex`, `Signed Decimal`, `Unsigned Decimal`. |

### Value Returned

| | |
|---|---|
| *o_bus* | Returns an output digital bus. |
| `nil` | Returns `nil` if there is an error. |

### Example

Following are the examples to create a digital binary bus with the 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")
```

## awvCreateBusFromWaveList

```
awvCreateBusFromWaveList(
     l_waveList
     )
     => o_bus | nil
```

### Description

Creates a digital bus from a list of digital waves provided as input.

### Arguments

| | |
|---|---|
| *l_wavelist* | List of digital waves or expressions (in string format), which yield digital waves. |

### Value Returned

| | |
|---|---|
| *o_bus* | A digital bus whose bits are the input digital waves. The first wave in the list corresponds to MSB and the last one corresponds to LSB. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvCreateBusFromWaveList( list(dig1 dig2 dig3 dig4 ) )
=> srrWave: 170938824
```

Creates a digital bus from the digital waves `dig1`, `dig2`, `dig3`, and `dig4`.

## awvCreatePlotWindow

```
awvCreatePlotWindow(
     [ ?parentWindow     w_windowId ]
     )
     => w_windowId | nil
```

### Description

Creates a Waveform window and returns the window ID.

### Arguments

| | |
|---|---|
| `?parentWindow` *`w_windowId`* | Waveform window ID for the parent window. If the parent window is a graphics editor window, the new Waveform window uses pens from the parent widow's technology file. |

### Value Returned

| | |
|---|---|
| *`w_windowId`* | Returns the ID for the new Waveform window. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvCreatePlotWindow( ?bBox list( 0:0 500:500 ) )
=> window:2
```

Creates a Waveform window that is 500 by 500 pixels with the lower left corner of the window in the lower left corner of the screen.

### Related Function

To delete a window, use the `hiCloseWindow` function or your window manager's close command. For more information about `hiCloseWindow`, see the *Cadence User Interface SKILL Reference*.

## awvLoadCustomCalcFunction

```
awvLoadCustomCalcFunction(
     [ ?fileName t_fileName ]
     [ ?funcName t_funcName ]
     [ ?templateFileName t_templateFileName ]
     [ @rest args ]
     )
     => t / nil
```

### Description

Loads the specified custom function template to the Function Panel of Calculator. This function template contains the UI definitions corresponding to the given SKILL function. These UI definitions can be obtained from the specified SKILL file or from a separate .ocn file.

### Arguments

| | |
|---|---|
| `?fileName` *`t_fileName`* | Name of the SKILL File in which the SKILL definitions of the custom function you want to load are defined. This SKILL file can also contain the UI templates for the functions defined in it. |
| `?funcName` *`t_funcName`* | Name of the function that you want to add to the Function Panel of the Calculator window. |
| `?templateFileName` *`t_templateFileName`* | Name of the .ocn file that defines the UI template for the custom function you want to load. |
| *`@rest args`* | Variable list of arguments passed to this function (as created from the Calculator UI).<br><br>**Note:** *`t_skillFilePath`* and *`t_ocnFilePath`* are added for backward compatibility only. It will be deprecated in future releases. |

### Values Returned

| | |
|---|---|
| `t` | Returns `t` when the custom function is successfully added. |
| `nil` | Returns `nil` if there is an error. |

### Examples

**Case 1**: Consider the following examples in which the UI template function is defined within the SKILL file:

- Adding the template for function, `func1`:

```
awvLoadCustomCalcFunction(?funcList "func1” ?fileName “test.il”)
```

```
awvLoadCustomCalcFunction(?funcList list("func1”) ?fileName
“test.il”)
```

- Adding the UI template function for func2 and func3:

```
awvLoadCustomCalcFunction(?funcList list(“func2” "func3"
?fileName “test.il” )
```

- Adding the UI template for all functions whose template functions are defined in 'test.il':

```
awvLoadCustomCalcFunction(?fileName "test.il")
```

**Case 2**: Consider the following example in which the UI template function for `func1` is obtained from a separate .ocn file, `fun1.ocn`:

```
awvLoadCustomCalcFunction(?funcList list("func1")  ?fileName
"test.il"  ?templateFileName "func1.ocn")
```

## awvLoadSharedCustomFunctionsFile

```
awvLoadSharedCustomFunctionsFile(
     t_fileName
     )
     => t / nil
```

### Description

This function is used to share custom functions template among multiple users from a central file location.

### Arguments

| | |
|---|---|
| *t_fileName* | Name of the custom functions template file that you want to share. |

### Values Returned

| | |
|---|---|
| `t` | Returns `t` when the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

### Example

The below example shows how to load the custom functions defined in `central.ini` file at a central location.

1. Specify the below functions to add two custom functions `Func1` and `Func2` in a file `central.ini`.

```
awvLoadCustomCalcFunction("Func1" "test1.il" "Func1.ocn")
awvLoadCustomCalcFunction("Fun2.ocn" "test2.il" "Func2.ocn")
```

2. Now, specify the below API to load the central.ini from your .cdsinit file. This will make these custom functions available at the central location and multiple users can access them.

```
awvLoadSharedCustomFunctionsFile("central.ini")
```

## awvDeleteAllWaveforms

```
awvDeleteAllWaveforms(
     w_windowId
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Deletes all the waveforms in the specified subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| t | Returns t when the waveforms are deleted from the subwindow. |
| nil | Return nil if the Waveform window or the subwindow do not exist. |

### Example

```
awvDeleteAllWaveforms( window(2) ?subwindow 4 )
=> t
```

Deletes all the waveforms in subwindow 4.

### Related Function

To delete a specific waveform curve from a subwindow, see the awvDeleteWaveform function on page-61.

## awvDeleteMarker

```
awvDeleteMarker(
     w_windowId
     t_bookmarkId | l_bookmarkIds
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

### Description

Deletes the specified bookmark(s) from the given window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *t_bookmarkId* | Bookmark ID or list of bookmark IDs to be deleted. |
| *x_subwindow* | Subwindow ID from which the bookmark is to be deleted. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the specified bookmarks are deleted from the subwindow. |
| `nil` | Return `nil` if the Waveform window or the subwindow do not exist. |

### Example

This example deletes the bookmark with ID=`bm` from the current window.

```
awvDeleteMarker(window bm)
```

## awvDeleteSubwindow

```
awvDeleteSubwindow(
     w_windowId
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Deletes a subwindow from a Waveform window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| t | Returns t when the subwindow is deleted. |
| nil | Returns nil if the Waveform window or the subwindow do not exist. |

### Example

```
awvDeleteSubwindow( window(3) ?subwindow 4 )
=> t
```

Removes subwindow 4 from the Waveform window.

### Related Function

To add a subwindow to a Waveform window, see the awvAddSubwindow function on page-32.

## awvDeleteWaveform

```
awvDeleteWaveform(
     w_windowId
     x_index
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Deletes a waveform curve from a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *x_index* | Integer identifying the waveform curve. |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| t | Returns t when the waveform is deleted. |
| nil | Returns nil if there is an error. |

### Example

```
awvDeleteWaveform( window(2) 1 ?subwindow 3 )
=> t
```

Deletes waveform curve 1 from subwindow 3 of a Waveform window.

### Related Function

To delete all the waveforms in a subwindow, see the awvDeleteAllWaveforms function on page-58.

## awvDigital2Analog

```
awvDigital2Analog(
     o_waveform
     n_vhi
     n_vlo
     s_VX
     [ ?mode s_mode ]
     [ ?outWaveType s_outWaveType ]
     [ ?vprevSTART s_vprevSTART ]
     )
     => o_waveform | nil
```

### Description

Computes the analog output of the provided digital waveform `o_waveform`.

### Arguments

| | |
|---|---|
| *o_waveform* | Represents the digital wave or bus that is to be converted to analog. This may be a simple wave or bus, a list of waves or buses, a string representing the expression yielding a wave or bus. |
| *o_vhi* | The high analog value to which the digital 1 (for single bit waveform) or maximum possible bus value is converted. |
| *o_vlo* | The low analog value to which the digital 0 (for single bit waveform) or minimum possible bus value is converted. |
| *s_VX* | The value to which state X of the digital wave is converted. It can be a number or a simple expression of `vhi`, `vlo` and `vprev` (that is, the previous value) in the form of a string. |
| `?mode` *s_mode* | A string to be provided if `t_waveform` is a bus or it is a list with at least one bus. Valid values: `wavelist`, `busvalue`<br>`wavelist`: the input bus will be converted into a list of analog waves, each representing a single bit digital waveform in the bus.<br>`busvalue`: the output will be a single analog wave representing the value of the bus. |
| `?outWaveType` *s_outWaveType* | A string to be provided if `t_waveform` is a bus or it is a list with at least one bus. Valid values: `pwl`, `zeroT`<br>`pwl`: the points in the outputted analog waveform will be joined by straight line segments.<br>`zeroT`: the output analog waveform will have voltage transitions in zero time. |
| `?vprevSTART` *s_vprevSTART* | The initial value of `vprev` to be used. This is required if the provided digital waveform starts at state X and `vprev` is used in the expression supplied in `s_VX`. This value overrides the value specified by the `.cdsenv` variable `vprevSTART` for the tool `calculator.d2`. |

### Value Returned

| | |
|---|---|
| *o_waveform* | A waveform is returned if the input was a single-bit digital waveform or if the input was a bus and the specified `s_mode` was `busvalue`. A list of waveforms is returned if the input was a list of single-bit digital waveforms or if the input was a bus and the specified `s_mode` was `wavelist`. |

| | |
|---|---|
| `nil` | Returns `nil` if the specified waveform window does not exist. |

### Example

```
awvDigital2Analog( bus1 5.0 0 "(vhi + vlo)/2.0" ?mode "busvalue" ?outWaveType
"zeroT")
```

Returns the analog, zero transition wave representing the specified digital bus `bus1`.

## awvDisableRedraw

```
awvDisableRedraw(
     w_windowId
     g_disable
     )
     => t / nil
```

### Description

Disables or enables redraw of the Waveform window based on the value of the *g_disable* flag. You might use this function to freeze the Waveform window display, send several plots to the window, then unfreeze the window to display all the plots at once.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *g_disable* | Flag that specifies whether the redraw feature of the window is disabled. Valid Values: `t` (redraw is disabled) or `nil` (redraw is enabled) |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when redraw is successfully disabled or enabled. |
| `nil` | Returns `nil` if there is an error, such as the specified Waveform window does not exist. |

## awvDisplayDate

```
awvDisplayDate(
     w_windowId
     )
     => t / nil
```

### Description

Displays the current date and time in the Waveform windowwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the date and time is displayed. |
| `nil` | Returns `nil` if the specified Waveform window does not exist. |

### Related Functions

To remove the date and time from a Waveform window, see the awvRemoveDate function on page-163.

To display a title in a subwindow, see the awvDisplaySubwindowTitle function on page-68.

To display a title on a Waveform window, see the awvDisplayTitle function on page-69.

## awvDisplayGrid

```
awvDisplayGrid(
     w_windowId
     g_on
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

### Description

Sets the display status for a grid on the indicated waveform subwindow.

### Arguments

| | |
|---|---|
| *w_window* | window ID |
| *g_on* | flag for turning the grid on (`t`) or off (`nil`) |
| `?subwindow` *x_subwindow* | subwindow number, defaults to 1 |

### Example

```
awvDisplayGrid(awvGetCurrentWindow() t) => t
```

## awvDisplaySubwindowTitle

```
awvDisplaySubwindowTitle(
     w_windowId
     t_title
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Displays a title in a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *t_title* | Title for the subwindow. |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| t | Returns t when the title is displayed. |
| nil | Returns nil if there is an error. |

### Example

```
awvDisplaySubwindowTitle( window(2) "Transient Response" ?subwindow 1 )
```

Displays *Transient Response* as the title for subwindow 1.

### Related Functions

To remove the title from a subwindow, see the awvRemoveSubwindowTitle function on page-166.

To display a title on a Waveform window, see the awvDisplayTitle function on page-69.

To remove the title from a Waveform window, see the awvRemoveTitle function on page-167.

## awvDisplayTitle

```
awvDisplayTitle(
     w_windowId
     t_title
     )
     => t / nil
```

### Description

Displays a title on a Waveform window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *t_title* | Title for the Waveform window. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the title is displayed. |
| `nil` | Returns `nil` if the specified Waveform window does not exist. |

### Example

```
awvDisplayTitle( window(2) "Transient Response" )
```

Displays *Transient Response* as the title for the Waveform window 2.

### Related Functions

To remove the title from a Waveform window, see the awvRemoveTitle function on page-167.

To display a title in a subwindow, see the awvDisplaySubwindowTitle function on page-68.

## awvEraseWindowMenuCB

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

### Description

Deletes all the objects (e.g. waveforms, markers) from the waveform subwindow.

The function is defined in `dfII/etc/context/awv.cxt`.

### Arguments

None

### Value Returned

| | |
|---|---|
| *t* | Returns `t` if the command was successful. |
| *nil* | Otherwise, returns `nil`. |

### Example

```
awvEraseWindowMenuCB() => t
```

## awvEval

```
awvEval(
     expr
     l_expr
     )
     => expr / nil
```

### Description

Returns the expression

### Arguments

| | |
|---|---|
| *expr* | Dummy argument used as the return value |
| *l_expr* | List of expressions to be evaluated. |

### Value Returned

| | |
|---|---|
| *expr* | Returns the first argument when function runs successfully. |
| `nil` | Returns `nil` otherwise. |

### Example

```
w = VT("/out")

awvEval("Expr_1" w)

This function returns w, which is the second argument.
```

## awvExitWindowFunctionAdd

```
awvExitWindowFunctionAdd(
     u_func
     )
     => t / nil
```

### Description

Adds a function to the list of functions that are called when you close a Waveform window or exit the Cadence software while a Waveform Window is open.

### Arguments

| | |
|---|---|
| *u_func* | Function to add to the list. Callback parameter list: (*w_windowId*) |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the function is added to the list. |
| `nil` | Returns `nil` otherwise. |

### Example

```
procedure( myExitFunc( windowId ) awvSaveWindow( windowId
    sprintf( nil "~/wave%d" windowId->windowNum ) )
)
awvExitWindowFunctionAdd( 'myExitFunc )
```

Adds a procedure called `myExitFunc` to the list of functions that are called when a Waveform window is closed. The `myExitFunc` function automatically saves the Waveform window.

### Related Functions

To delete a function from the list of *exit* functions, see the awvExitWindowFunctionDel function on page-73.

To get the list of *exit* functions, see the awvExitWindowFunctionGet function on page-74.

## awvExitWindowFunctionDel

```
awvExitWindowFunctionDel(
     u_func
     )
     => t | nil
```

### Description

Deletes a function from the list of functions that are called when you close a Waveform window or exit the Cadence software while a Waveform window is open.

### Arguments

| | |
|---|---|
| *u_func* | Function to delete from the list. Callback parameter list: (*w_windowId*) |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the function is deleted from the list. |
| `nil` | Returns `nil` otherwise. |

### Example

```
awvExitWindowFunctionDel( 'myExitFunc )
=> t
```

Removes `myExitFunc` from the list of exit functions.

### Related Functions

To add an exit function, see the awvExitWindowFunctionAdd function on page-72.

To get the list of *exit* functions, see the awvExitWindowFunctionGet function on page-74.

## awvExitWindowFunctionGet

```
awvExitWindowFunctionGet()
     => l_initFunctionList | nil
```

### Description

Gets the list of functions that are called when you close a Waveform window or exit the Cadence software while a Waveform window is open.

### Arguments

None.

### Value Returned

| | |
|---|---|
| *l_initFunctionList* | Returns the list of functions. |
| `nil` | Returns `nil` if the list is empty. |

### Example

```
awvExitWindowFunctionGet
=> (myExitFunc)
```

Returns a list containing the names of the exit functions for Waveform Windows. In this case, there is only one exit function.

### Related Functions

To add an `exit` function, see the awvExitWindowFunctionAdd function on page-72.

To delete a function from the list of `exit` functions, see the awvExitWindowFunctionDel function on page-73.

## awvGetAssertName

```
awvGetAssertName(
     o_waveform
     )
     => t_assertName / nil
```

### Description

Returns the name of assert defined in violation data.

### Arguments

| | |
|---|---|
| *o_waveform* | Waveform object. |

### Values Returned

| | |
|---|---|
| *t_assertName* | Returns the name of assert when the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
openResults("psf")
// Opens the results directory in psf format.
selectResults 'asserts
// Select the type of results
w = getData("check1.I3.I32.M1.violation1")
awvGetAssertName(w)
("check1")
//Name of the assert
```

## awvEyeCross

```
awvEyeCross(
     w_waveform
     n_start
     n_stop
     n_period
     n_threshold
     [ x_edgeType ]
     [ x_ignoreStart ]
     [ x_ignoreEnd ]
     )
     => o_waveform / nil
```

### Description

Returns an output waveform showing X-axis values where the given eye diagram crosses the specified threshold on Y-axis.

### Arguments

| | |
|---|---|
| *w_waveform* | The eye diagram waveform. |
| *n_start* | Start time of the time signal. |
| *n_stop* | Stop time of the time signal. |
| *n_period* | Eye period of the specified eye diagram. |
| *n_threshold* | The threshold value. |
| *x_edgeType* | (Optional) Type of cross<br>Valid values: `"rising" "falling"`, or `"either"`<br>Default value: `"either"` |
| *x_ignoreStart* | (Optional) The time before which any crosses occurred in the eye diagram will be ignored. |
| *x_ignoreEnd* | (Optional) The time after which any crosses occurred in the eye diagram will be ignored. |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns an output waveform showing cross number on X-axis and cross time on Y-axis. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider the following eye diagram created from a jitter signal with time from `1u` to `20u` and an eye period of `80n`.

```
eye = eyeDiagram(v("jitter" ?result "tran-tran") 1u 20u 80n)
```

- If you want to return the waveform representing X-axis values between `1u` (start) and `20u` (stop) where the specified eye diagram crosses the threshold value `0.5` on Y-axis, run the `awvCrossEye` function as shown below:

  ```
  awvEyeCross(eye 1u 20u 80n 0.5)
  ```

- If you want to return crosses of any type between `0` and `40n`, run the `awvCrossEye` function as shown below:

  ```
  awvEyeCross(eye 1u 20u 80n 0.5 "either" 0 40n)
  ```

- If you want to return crosses of any type between `40n` and `80n`, run the `awvCrossEye` function as shown below:

```
awvEyeCross(eye 1u 20u 80n 0.5 "either" 40n 80n)
```

## awvGetCurrentSubwindow

```
awvGetCurrentSubwindow(
     w_windowId
     )
     => x_subwindow / nil
```

### Description

Returns the current subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |

### Value Returned

| | |
|---|---|
| *x_subwindow* | Returns the identification number found in the upper right corner of the current subwindow. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvGetCurrentSubwindow( window(2) )
=> 3
```

Returns 3 as the current subwindow for Waveform window 2.

### Related Function

To specify a subwindow as the current subwindow, see the awvSetCurrentSubwindow function on page-179.

## awvGetCurrentWindow

```
awvGetCurrentWindow()
     => w_windowId | nil
```

### Description

Returns the window ID for the current Waveform window.

### Arguments

None.

### Value Returned

| | |
|---|---|
| *w_windowId* | Returns the ID for the current Waveform window. |
| nil | Returns nil if there is no current Waveform window. |

### Example

```
awvGetCurrentWindow
=> window:4
```

Returns *window:4* as the current window.

### Related Function

To specify a Waveform window as the current window, see the awvSetCurrentWindow function on page-180.

## awvGetDisplayMode

```
awvGetDisplayMode(
     w_windowId
     [ ?subwindow   x_subwindow ]
     )
     => t_mode | nil
```

### Description

Returns the display mode of a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| *t_mode* | Returns the display mode of the subwindow (`strip`, `smith`, or `composite`). |
| `nil` | Returns `nil` if there is an error. |

### Related Functions

To set the display mode of a subwindow, see the awvSetDisplayMode function on page-182.

To return the Smith display type of a subwindow, see the awvGetSmithModeType function on page-91.

To set the Smith display mode for a subwindow, see the awvSetSmithModeType function on page-190.

## awvGetDrawStatus

```
awvGetDrawStatus(
     w_windowId
     )
     => t / nil
```

### Description

Returns the draw status of the waveform display window.

### Arguments

| | |
|---|---|
| *w_windowId* | window Id |

### Example

```
awvGetDrawStatus(awvGetCurrentWindow()) => t
```

## awvGetHiWindow

```
awvGetHiWindow(
     w_windowId
     )
     => w_HIWindowId/nil
```

### Description

Returns the ID of the waveform HI window corresponding to the specified waveform window ID. This function is primarily used to support menu customization using HI calls.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |

### Value Returned

| | |
|---|---|
| *w_HIWindowId* | Returns the HI window ID. |
| nil | Returns nil if there is an error. |

### Examples

- `win = awvGetHiWindow(windowId)`

  Here, `windowId` is the ID of the waveform window corresponding to which you want to return the HI window ID.

- `awvGetHiWindow(awvGetCurrentWindow())`

  Here, the ID of the current waveform window is obtained using the `awvGetCurrentWindow` function.

- Consider the following example in which you create a new menu, *Test*, with an option, *Item1*. When you click *Item1*, the 'Hello' message is displayed in CIW.

```
windowId=awvGetHiWindow(awvGetCurrentWindow())
procedure( myCreateMenu( windowId)
     win = awvGetHiWindow(windowId)
     item1 = hiCreateMenuItem( ?name 'item1 ?itemText "Item1" ?callback
"println(\"Hello\")")
     hiCreatePulldownMenu( 'myMenu "Test" list(item1))
     hiInsertBannerMenu(win myMenu hiGetNumMenus(win))
```

```
)
awvInitWindowFunctionAdd('myCreateMenu)
```

## awvGetInitializationTimeout

```
awvGetInitializationTimeout()
     => x_timeOut
```

### Description

Retrieves the time-out period (in seconds) set for ADE to establish connection with Virtuoso Visualization and Analysis XL.

### Arguments

None.

### Value Returned

| | |
|---|---|
| *x_timeOut* | Time-out period (in seconds). |

### Example

```
awvGetInitializationTimeout()
    => 240
```

This means that the time-out period for ADE to establish connection with Virtuoso Visualization and Analysis XL was set to 240 `seconds`.

## awvGetOnSubwindowList

```
awvGetOnSubwindowList(
     w_windowId
     [ ?all    g_all ]
     )
     => l_onSubwindows | nil
```

### Description

Returns the list of subwindows that are being used in the specified Waveform window. This list includes only subwindows whose display and update statuses are turned *on*. To get a list of all subwindows whose displays are *on*, regardless of update status, set `g_all` to `t`.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?all *g_all* | Boolean flag that specifies whether the list shall contain every subwindow whose display is *on*, regardless of update status. Valid Values: `t` to include all subwindows whose displays are *on*, regardless of update status, or `nil` to include only subwindows whose update statuses and display are *on*. |

### Value Returned

| | |
|---|---|
| *l_onSubwindows* | Returns the list of subwindows whose displays are *on*. |
| nil | Returns `nil` if the specified Waveform window does not exist. |

### Example

```
awvGetOnSubwindowList( window(5) ?all t )
=> (1 2 3)
```

Returns a list of three subwindows that are being used in Waveform window 5, including subwindows whose update statuses are *off*.

### Related Function

To get a list of all the subwindows whose update statuses are turned *on* (regardless of whether their displays are *on* or *off*), see the awvGetSubwindowList function on page-95.

## awvGetPlotStyle

```
awvGetPlotStyle(
     w_windowId
     [ ?subwindow   x_subwindow ]
     )
     => s_style
```

### Description

Gets the plotting style for the waveforms in a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *x_subwindow* | Number for the subwindow (found in the upper right corner).<br>Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| *s_style* | Returns the plotting style for the subwindow. |

### Example

```
awvGetPlotStyle( window(4) ?subwindow 2 )
=> scatterPlot
```

Returns `scatterPlot` as the plotting style for subwindow 2.

### Related Function

To set the plotting style for all the waveforms in a subwindow, see the awvSetPlotStyle function on page-189.

## awvGetScalarFromWave

```
awvGetScalarFromWave(
     o_waveform
     )
     => n_yValue / o_waveform
```

### Description

Returns the Y-axis value of the point when the input waveform is single point. If the waveform has multiple points, this function returns back the input waveform.

### Arguments

| | |
|---|---|
| `o_waveform` | Input waveform |

### Values Returned

| | |
|---|---|
| `n_yValue` | Returns the Y-axis value of the point in single-point waveform. |
| `o_waveform` | Returns the input waveoform if it has multiple points. |

### Examples

Consider the following examples:

- When the input is a single-point waveform, the `awvGetScalarFromWave` function returns the y-value of the single point in the waveform.

```
xVec = drCreateVec('double 1)
yVec = drCreateVec('double 1)
drSetElem(xVec 0 1)
drSetElem(yVec 0 1)
wave = drCreateWaveform(xVec yVec)
awvGetScalarFromWave(wave)
==> 1.0
```

- When the input is a scalar, the `awvGetScalarFromWave` function returns back the scalar:

```
awvGetScalarFromWave(0.5)
==> 0.5
```

- When the input waveform has multiple points, the `awvGetScalarFromWave` function returns back the waveform.

```
xVec = drCreateVec('double 10)
yVec = drCreateVec('double 10)
for(i 1 10
  drSetElem(xVec i i)
  drSetElem(yVec i i*i-19*i+100)
)
wave = drCreateWaveform(xVec yVec)
awvGetScalarFromWave(wave)
==> wave
```

## awvGetSelectedTraceWaveforms

```
awvGetSelectedTraceWaveforms(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => l_waveformList / nil
```

### Description

Returns a list of waveforms corresponding to the traces that are selected in the specified or current subwindow of the given window.

**Note:** This function does not support trace groups.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| *l_waveformList* | Returns a list of waveforms corresponding to the selected traces. |
| `nil` | Returns `nil` if no traces are selected or there is an error. |

### Example

```
awvGetSelectedTraceWaveforms(awvGetCurrentWindow() ?subwindow 1)
=> (srrWave:0x08e59020)
awvGetSelectedTraceWaveforms(awvGetCurrentWindow() ?subwindow 1)
=> nil (in case no traces are selected)
```

## awvGetSmithModeType

```
awvGetSmithModeType(
     w_windowId
     [ ?subwindow   x_subwindow ]
     )
     => t_type / nil
```

### Description

Returns the Smith display type of a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| *t_type* | Returns the type of Smith display (`impedance`, `admittance`, or `polar`). |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvGetSmithModeType( window(2) ?subwindow 3)
=> "polar"
```

Returns polar as the Smith display type for subwindow 3.

### Related Functions

To set the Smith display type of a subwindow, see the awvSetSmithModeType function on page-190.

To return the display mode of a subwindow, see the awvGetDisplayMode function on page-81.

To set the display mode of a subwindow, see the awvSetDisplayMode function on page-182.

## awvGetStripNumberOfSelectedTrace

```
awvGetStripNumberOfSelectedTrace(
     w_windowId
     )
     => n_stripNumber / nil
```

### Description

Returns the strip number of the selected trace plotted in the specified window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |

### Value Returned

| | |
|---|---|
| *n_stripNumber* | Returns the strip number of the selected trace. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
W00 = drCreateWaveform( drCreateVec( 'double list( 1 2 3 4 5 )) drCreateVec( 'double
list( 10 20 30 40 50 )))
W01 = drCreateWaveform( drCreateVec( 'double list( 2 4 6 8 9 )) drCreateVec( 'double
list( 2 4 8 16 32 )))
plot W00
plot W01
window=awvGetCurrentWindow()
awvGetStripNumberOfSelectedTrace(window)"
```

Returns the strip number of the selected trace plotted in the current window.

## awvGetStripNumbersList

```
awvGetStripNumbersList(
     w_windowId
     [ ?subwindow   x_subwindow ]
     )
     => l_stripNumberList | nil
```

### Description

Returns the list of strip numbers in the specified subwindow of the given window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?subwindow<br>*x_subwindow* | Number for the subwindow (found in the upper right corner).<br>Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| *l_stripNumberList* | Returns the list of strip numbers in the specified subwindow. |
| nil | Returns nil if there is an error. |

### Example

```
W00 = drCreateWaveform( drCreateVec( 'double list( 1 2 3 4 5 )) drCreateVec( 'double
list( 10 20 30 40 50 )))
W01 = drCreateWaveform( drCreateVec( 'double list( 2 4 6 8 9 )) drCreateVec( 'double
list( 2 4 8 16 32 )))
plot W00
plot W01
window=awvGetCurrentWindow()
awvGetStripNumbersList(window)
// with Subwindow:
awvGetStripNumbersList(window ?subwindow 1)
```

Returns the list of strip numbers in the current window.

## awvGetSubwindowStripCount

```
awvGetSubwindowStripCount(
     w_windowId
     [ ?subwindow   x_subwindow ]
     )
     => n_stripNumberCount | nil
```

### Description

Returns the total number of strips displayed in the specified subwindow of the given window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| *l_stripNumberCount* | Returns the total number of strip numbers displayed in the specified subwindow. |
| nil | Returns nil if there is an error. |

### Example

```
W00 = drCreateWaveform( drCreateVec( 'double list( 1 2 3 4 5 )) drCreateVec( 'double
list( 10 20 30 40 50 )))
W01 = drCreateWaveform( drCreateVec( 'double list( 2 4 6 8 9 )) drCreateVec( 'double
list( 2 4 8 16 32 )))
plot W00
plot W01
window=awvGetCurrentWindow()
awvGetSubwindowStripCount(window)
//with subwindow:
awvGetSubwindowStripCount(window ?subwindow 1)
```

Returns the number of strips in the current subwindow.

## awvGetSubwindowList

```
awvGetSubwindowList(
     w_windowId
     [ ?all    g_all ]
     )
     => l_subwindows | nil
```

### Description

Returns a list of all the subwindows whose update statuses are turned *on*, regardless of whether their displays are *on* or *off*. To get a list of all subwindows, including subwindows whose update statuses are *off*, set `g_all` to `t`.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?all *g_all* | Boolean flag that specifies whether the function returns all subwindows, or only subwindows whose update statuses are turned *on*. Valid Values: `t` to return all subwindows, `nil` to return only updatable subwindows. |

### Value Returned

| | |
|---|---|
| *l_subwindows* | Returns the list of all the subwindows for a Waveform window. |
| nil | Returns `nil` if the specified Waveform window does not exist. |

### Example

```
awvGetSubwindowList( window(5) ?all t )
=> (1 2 3 4)
```

Returns a list of all the subwindows for Waveform window 5, including subwindows whose update statuses are turned *off*.

### Related Function

To return a list of subwindows that are being used in a Waveform window (including *only* subwindows whose display and update statuses are turned *on*), see the awvGetOnSubwindowList function on page-86.

## awvGetUnusedEntityList

```
awvGetUnusedEntityList(
     w_windowId
     [ ?subwindow   x_subwindow ]
     [ ?total  x_total ]
     )
     => l_waveformEntityIndices | nil
```

### Description

Returns a list of integers that have not already been used to identify curves in a subwindow. You can specify the total number of integers to include in the return value with *x_total*.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |
| ?total *x_total* | Specifies the number of unused curves to include in the return value, which is a list. Default Value: 20 |

### Value Returned

| | |
|---|---|
| *l_waveformEntityIndices* | Returns a list of the next lowest integers that have not been used to identify existing curves. |
| nil | Returns nil if there is an error. |

### Example

```
awvGetUnusedEntityList( window(2) ?subwindow 3 ?total 10 )
=>
(6 7 8 9 10
     11 12 13 14 15
)
```

Returns numbers 6 through 15 as the next 10 unused numbers. This means that curves 1 through 5 already exist in the window.

## awvGetWaveNameList

```
awvGetWaveNameList(
     w_windowId
     [ ?subwindow   x_subwindow ]
     )
     => l_infoList
```

### Description

Returns a list that contains two elements. The first element is a list of numbers for the curves and the second is a list of the corresponding names.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?subwindow<br>*x_subwindow* | Number for the subwindow (found in the upper right corner).<br>Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| *l_infoList* | Returns a list with the numbers and another list with the names for the waveform curves. |

### Example

```
awvGetWaveNameList( window(3) ?subwindow 1)
=>
((2 3)
("/net30" "/net50")
)
```

Returns a list with the numbers and a list with the names for waveform curves 2 and 3.

## awvGetWindowList

```
awvGetWindowList()
     => l_windows
```

### Description

Returns a list of all the Waveform Windows associated with the current process.

### Arguments

None.

### Value Returned

| | |
|---|---|
| *l_windows* | Returns a list of all the Waveform Windows. |

### Example

```
awvGetWindowList
=> ( window:3 window:4 )
```

Returns the identifiers for two Waveform Windows.

## awvGetXAxisLabel

```
awvGetXAxisLabel(
     w_windowId
     [ ?subwindow   x_subwindow ]
     [ ?computed    g_computed ]
     )
     => t_label | nil
```

### Description

Returns the user-specified X axis label if you set `computed` to `nil`. Returns the system-computed X axis label otherwise.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?subwindow<br>*x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |
| *g_computed* | Boolean flag that specifies whether to return the user-specified X axis label or the system-computed X axis label. Valid Values: `t`, which specifies that the system-computed X axis label is returned, or `nil`, which specifies that the user-defined X axis label is returned. |

### Value Returned

| | |
|---|---|
| *t_label* | Returns the X axis label for the specified subwindow. |
| nil | Returns `nil` if there is an error. |

### Example

```
awvGetXAxisLabel( window(7) ?subwindow 1 ?computed nil)
=> "seconds"
```

Returns *seconds* as the user-defined X axis label for subwindow 1.

### Related Functions

If you want to specify an X axis label to replace the automatically computed label, see the awvSetXAxisLabel function on page-200.

To return the Y axis label, see the awvGetYAxisLabel function on page-102.

If you want to specify a Y axis label to replace the automatically computed label, see the awvSetYAxisLabel function on page-203.

## awvGetXMarkerNames

```
awvGetXMarkerNames(
     w_windowId
     [ ?subwindow   x_subwindow ]
     )
     => l_markerNames | nil
```

### Description

Returns the names of all the X markers in a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?subwindow<br>*x_subwindow* | Number for the subwindow (found in the upper right corner).<br>Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| *l_markerNames* | Returns a list of all the X markers in the subwindow. |
| nil | Returns nil if there is an error. |

### Example

```
awvGetXMarkerNames( window(2) ?subwindow 1 )
=> ( "M1" "M2" "M3" )
```

Returns the names of the X markers in subwindow 1.

### Related Function

To return the names of all the Y markers in a subwindow, see the awvGetYMarkerNames function on page-104.

## awvGetYAxisLabel

```
awvGetYAxisLabel(
     w_windowId
     x_yNumber
     [ ?subwindow   x_subwindow ]
     [ ?computed    g_computed ]
     [ ?stripNumber x_stripNumber ]
     )
     => t_label | nil
```

### Description

Returns the user-specified Y axis label if you set `computed` to `nil`. Returns the system-computed Y axis label otherwise.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *x_yNumber* | Specifies the Y axis whose label you want to get. Valid Values: 1 through 4 |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |
| ?computed *g_computed* | Boolean flag that specifies whether to return the user-specified Y axis label or the system-computed Y axis label. Valid Values: `t`, which specifies that the system-computed Y axis label is returned, or `nil`, which specifies that the user-defined Y axis label is returned. |
| ?stripNumber *x_stripNumber* | Specifies the strip containing the Y axis. Valid Values: 1 through 20 |

### Value Returned

| | |
|---|---|
| *t_label* | Returns the Y axis label for the specified subwindow. |
| nil | Returns `nil` if there is an error. |

### Example

```
awvGetYAxisLabel( window(4) 2 ?subwindow 3 ?computed nil)
=> "Voltage"
```

Returns `Voltage` as the user-defined Y axis label for Y axis 2 of subwindow 3.

### Related Functions

If you want to specify a Y axis label to replace the automatically computed label, see awvSetYAxisLabel.

To return the X axis label, see awvGetXAxisLabel.

If you want to specify an X axis label to replace the automatically computed label, see awvSetXAxisLabel.

## awvGetYMarkerNames

```
awvGetYMarkerNames(
     w_windowId
     [ ?subwindow   x_subwindow ]
     )
     => l_markerNames | nil
```

### Description

Returns the names of all the Y markers in a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?subwindow<br>*x_subwindow* | Number for the subwindow (found in the upper right corner).<br>Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| *l_markerNames* | Returns a list of all the Y markers in the subwindow. |
| nil | Returns nil if there is an error. |

### Example

```
awvGetYMarkerNames( window(2) ?subwindow 2 )
=> ( "M1" "M2" "M3" )
```

Returns the names of the Y markers in subwindow 2.

### Related Function

To return the names of all the X markers in a subwindow, see the awvGetXMarkerNames function on page-101.

## awvIsnitWindowFunctionAdd

```
awvInitWindowFunctionAdd(
     u_func
     )
     => t / nil
```

### Description

Adds a function to the list of functions that are called when a new Waveform window is opened. The list of functions is empty by default. For example, you can use this function to add menus to every new Waveform window that is opened.

### Arguments

| | |
|---|---|
| *u_func* | Function to add to the list. Callback parameter list: (*w_windowId*) |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the function is added to the list. |
| `nil` | Returns `nil` otherwise. |

### Example

```
procedure( myCreateMenu( windowId )
    let( ( item1 item2 )
        item1 = hiCreateMenuItem(
            ?name 'item1
            ?itemText "Item1"
            ?callback "myItem1CB()"
        )

        item2 = hiCreateMenuItem(
            ?name 'item2
            ?itemText "Item2"
            ?callback "myItem2CB()"
        )

        hiCreatePulldownMenu(
            'myMenu
            "Test"
            list( item1 item2 )
        )

        hiInsertBannerMenu( window myMenu hiGetNumMenus(window))
    )
```

```
)
awvInitWindowFunctionAdd( 'myCreateMenu )
```

Adds a procedure called `myCreateMenu`, which adds a banner menu called `Test` with two menu choices called *Item1* and *Item2*.

### Related Functions

To delete a function from the list of initialization functions, see the awvInitWindowFunctionDel function on page-107.

To get the list of initialization functions, see the awvInitWindowFunctionGet function on page-108.

## awvInitWindowFunctionDel

```
awvInitWindowFunctionDel(
     u_func
     )
     => t / nil
```

### Description

Deletes a function from the list of functions that are called when a new Waveform window is opened.

### Arguments

| | |
|---|---|
| *u_func* | Function to delete from the list. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the function is deleted from the list. |
| `nil` | Returns `nil` otherwise. |

### Example

```
awvInitWindowFunctionDel( 'myCreateMenu )
=> t
```

Removes the `myCreateMenu` function from the list of initialization functions.

### Related Functions

To add a function to the list of initialization functions, see the awvIsnitWindowFunctionAdd function on page-105.

To get the list of initialization functions, see the awvInitWindowFunctionGet function on page-108.

## awvInitWindowFunctionGet

```
awvInitWindowFunctionGet()
     => l_initFunctionList
```

### Description

Returns the current list of functions that are called when a new Waveform window is opened. This list is empty by default.

### Arguments

None.

### Value Returned

| | |
|---|---|
| *l_initFunctionList* | Returns the list of functions that are called when a new Waveform window is opened. |

### Example

```
awvInitWindowFunctionGet
=> (myCreateMenu)
```

Returns a list containing the names of the initialization functions for Waveform Windows. In this case, there is only one initialization function.

### Related Functions

To add a function to the list of initialization functions, see the awvIsnitWindowFunctionAdd function on page-105.

To delete a function from the list of initialization functions, see the awvInitWindowFunctionDel function on page-107.

## awvIsPlotWindow

```
awvIsPlotWindow(
     w_windowId
     )
     => t / nil
```

### Description

Returns `t` if the specified window is a Waveform window.

### Arguments

| | |
|---|---|
| *w_windowId* | Window ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the specified window is a Waveform window. |
| `nil` | Returns `nil` otherwise. |

### Example

```
awvIsPlotWindow( window(8))
=> t
```

Returns `t`, indicating that *window(8)* is a Waveform window.

## awvLoadEyeMask

```
awvLoadEyeMask(
     [ ?fileName t_fileName ]
     )
     => t / nil
```

### Description

Loads the eye mask saved in a VCSV file and adds it to the eye mask list in the Eye Diagram assistant. The loaded eye mask is displayed in the *Mask* drop-down list of the *Eye Mask* tab.

### Arguments

| | |
|---|---|
| `?fileName` *`t_fileName`* | Path of the VCSV file from which you want to load the eye mask. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the specified file is successfully loaded. |
| `nil` | Returns `nil` otherwise. |

### Example

```
awvLoadEyeMask(?fileName "../myMask.vcsv")
```

Adds the eye mask saved in the specified VCSV file, `myMask.vcsv`, to the *Mask* drop-down list.

## awvLoadMenuCB

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

### Description

Displays the *Load Menu (Windows -> Load ...)*.

The function is defined in `dfII/etc/context/awv.cxt`.

### Arguments

None

### Value Returned

| | |
|---|---|
| *t* | Returns `t` if the command was successful. |
| *nil* | Otherwise, returns `nil`. |

### Example

```
awvLoadMenuCB() => t
```

## awvLoadWindow

```
awvLoadWindow(
     w_windowId
     t_fileName
     [ ?resultsDir t_resultsDir ]
     )
     => t / nil
```

### Description

Initializes the state of a Waveform window from information saved in a file.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID for the window to be affected. |
| *t_fileName* | Name of the file containing the state of the Waveform window. |
| ?resultsDir *t_resultsDir* | Directory containing the PSF files (results). Remember to put quotation marks before and after the path name. |

### Value Returned

| | |
|---|---|
| t | Returns t when the Waveform window is initialized. |
| nil | Returns nil if there is an error. |

### Example

```
awvLoadWindow( window(3) "my_file")
=> t
```

Initializes Waveform window 3 using the information saved in *my_file* (in the current directory).

### Related Function

To save the state of a Waveform window, see the awvSaveWindowImage function on page-174.

## awvLogYAxis

```
awvLogYAxis(
     w_windowId
     x_yNumber
     g_state
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Sets the Y axis for a strip in a subwindow to display logarithmically if the *g_state* flag is set to `t`. If *g_state* is not set to `t`, the display is set to linear. If you do not specify a strip, the limits are applied when the Waveform window is in the *composite* mode.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *x_yNumber* | Specifies the Y axis whose display is to be set. Valid Values: 1 through 4 |
| *g_state* | Flag that specifies whether the display is logarithmic or linear. Valid Values: `t` (specifies that the display is logarithmic), or `nil` (specifies that the display is linear) |
| `?stripNumber` *x_stripNumber* | Specifies the strip in which the Y axis display is to be set. Valid Values: 1 through 20 |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the display of the Y axis is set. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvLogYAxis( window( 2 ) 1 t )
```

Sets the Y axis (Y1) to display logarithmically. This takes effect only in the *composite* mode.

```
awvLogYAxis( window( 2 ) 1 nil ?stripNumber 2 )
```

Sets the Y axis (Y1) in strip 2 to display linearly. This takes effect only in the *strip* mode.

### Related Function

To set the display mode of the X axis in a subwindow, see the awvSetXScale function on page-202.

## awvLogXAxis

```
awvLogXAxis(
     w_windowId
     g_state
     [ ?subwindow x_subwindowID ]
     )
     => t / nil
```

### Description

Sets the X axis for a strip in a subwindow to display logarithmically if the *g_state* flag is set to t. If *g_state* is not set to `t`, the display is set to linear.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window identifier |
| *g_state* | Flag that specifies whether the display is logarithmic or linear. Specifies `t` for log, otherwise sets it to the linear axis. The state set will not take effect if the axis type is *smith*. |
| `?subwindow` *x_subwindowId* | Subwindow number |

### Values Returned

| | |
|---|---|
| `t` | Returns t if the command was successful |
| `nil` | Returns nil if the command was successful. |

### Example

```
awvLogXAxis( window(2) t ?subwindow 2) => t
```

where `window(2)` corresponds to a waveform window.

## awvPlaceAMarker

```
awvPlaceAMarker(
     w_window
     x_traceIndex
     n_xLoc
     n_yLoc
     [ ?subwindow x_subwindow ]
     [ ?positionMode t_positionMode ]
     )
     => t_markerId / nil
```

### Description

Places marker of type A on the specified trace.

### Arguments

| | |
|---|---|
| *w_window* | Waveform window identifier. |
| *x_traceIndex* | Integer identifying the waveform curve. For more information about the index values, see awvGetWaveNameList.<br><br>It is mandatory to input an index value, otherwise, an error message is displayed. |
| *n_xLoc* | The x-coordinate at which the marker is to be placed. |
| *n_yLoc* | The y-coordinate at which the marker is to be placed. |
| ?subwindow *x_subwindow* | Subwindow number. |
| ?positionMode *t_positionMode* | Marker positioning mode.<br>Valid values: x, y, and xy<br>Default value: x<br><br>x— Marker is created at the specified *xLoc* value and the nearest *yLoc* value on the trace.<br>y— Marker is created at the specified *yLoc* value and the nearest *xLoc* value on the trace.<br>byXYMode— Marker is created at the specified *xLoc* and *yLoc* values on the trace. |

### Values Returned

| | |
|---|---|
| *t_markerId* | Returns the ID of the marker after the marker is placed on the specified trace. |
| nil | Returns nil if the command was unsuccessful. |

### Example

Consider the following example where you create a trace for the out signal and then place an A marker on this trace at the specified x and y locations.

```
win = awvCreatePlotWindow()
openResults("../ampsim.raw")
awvPlotWaveform(win list(v("out" ?result "tran-tran") v("net10" ?result "tran-
tran")))
awvGetWaveNameList(win)
```

This returns the index number of the waveforms. Now, run the `awvPlaceAMarker` function to place A marker on this trace:

```
awvPlaceAMarker( win 2 100n 0)
awvPlaceAMarker( win 2 100n 0 ?positionMode "x")
//Places the marker at the specified xLoc=2 and nearest YLoc on the trace.
awvPlaceAMarker( win 2 200n 1.5 ?positionMode "y")
//Places the marker at the specified yLoc=200 and nearest xLoc on the trace.
awvPlaceAMarker( win 2 30n 0 ?positionMode "xy")
//Places the marker at the specified x and Y locations on the trace.
```

## awvPlaceBMarker

```
awvPlaceBMarker(
     w_window
     x_traceIndex
     n_xLoc
     n_yLoc
     [ ?subwindow x_subwindow ]
     [ ?positionMode t_positionMode ]
     )
     => t_markerId / nil
```

### Description

Places marker of type B on the specified trace.

### Arguments

| | |
|---|---|
| *w_window* | Waveform window identifier. |
| *x_traceIndex* | Integer identifying the waveform curve. For more information about the index values, see awvGetWaveNameList.<br><br>It is mandatory to input an index value, otherwise, an error message is displayed. |
| *n_xLoc* | The x-coordinate at which to place the marker. |
| *n_yLoc* | The y-coordinate at which to place the marker. |
| ?subwindow *x_subwindow* | Subwindow number |
| ?positionMode *t_positionMode* | Marker positioning mode.<br>Valid values: x, y and xy<br>Default value: x<br><br>x— Marker is created at the specified *xLoc* value and the nearest *yLoc* value on the trace.<br>y— Marker is created at the specified *yLoc* value and the nearest *xLoc* value on the trace.<br>byXYMode— Marker is created at the specified *xLoc* and *yLoc* values on the trace.. |

### Values Returned

| | |
|---|---|
| *t_markerId* | Returns the ID of the marker after the marker is placed on the specified trace. |
| nil | Returns nil if the command was successful. |

### Example

Consider the following example where you create a trace for the out signal and then place an B marker on this trace at the specified x and y locations.

```
win = awvCreatePlotWindow()
openResults("../ampsim.raw")
awvPlotWaveform(win list(v("out" ?result "tran-tran") v("net10" ?result "tran-tran")))
awvGetWaveNameList(win)
```

This returns the index number of the waveforms. Now, run the `awvPlaceBMarker` function to place B marker on this trace:

```
awvPlaceBMarker( win 2 100n 0)
awvPlaceBMarker( win 2 100n 0 ?positionMode "x")
//Places the marker at the specified xLoc=2 and nearest YLoc on the trace.
awvPlaceBMarker( win 2 200n 1.5 ?positionMode "y")
//Places the marker at the specified yLoc=200 and nearest xLoc on the trace.
awvPlaceBMarker( win 2 30n 0 ?positionMode "xy")
//Places the marker at the specified x and Y locations on the trace.
```

## awvPlaceBookmark

```
awvPlaceBookmark(
     w_windowId
     t_bmType
     l_location
     [ ?waveIndex x_waveIndex ]
     [ ?parent t_parentGroup ]
     [ ?description t_description ]
     [ ?visible g_visible ]
     [ ?subwindow x_subwindow ]
     [ ?properties l_proplist ]
     )
     => l_bookmarkId / nil
```

### Description

Adds a bookmark in the specified window. You can create bookmarks on the graph for the regions that you are interested in. The bookmarks can be organized into groups, where a bookmark group can contain other groups.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *t_bmType* | Type of the bookmark to be created. The supported bookmark types are: `region` (Region Bookmark) , `xrange` (X Range Bookmark), `yrange` (Y Range Bookmark), and `point` (Point Bookmark). |
| *l_location* | For a `region` bookmark, includes a list of four coordinates that describes the location for bookmark.<br><br>For `xrange`, `yrange` and `point` bookmarks, includes a list of two waveform coordinates that describe the location for the bookmark.<br><br>Some Examples:<br><br>■ region: `awvPlaceBookmark(window "region" list(100.0ns 0 300.0ns 3) ?waveIndex 1 ?description "region_bookmark " ?visible t)`<br><br>■ xrange: `awvPlaceBookmark(window "xrange" list(100.0ns 400.0ns) ?waveIndex 1 ?description "xrange_bookmark" ?visible t)`<br><br>■ yrange: `awvPlaceBookmark(window "yrange" list(-0.8 4.0) ?waveIndex 1 ?description "yrange_bookmark"  ?visible t)`<br><br>■ point: `awvPlaceBookmark(window "point" list(125.0ns 3.5) ?waveIndex 1 ?description "point_bookmark" ?visible t ?subwindow 1)` |
| *x_waveIndex* | Integer identifying the waveform curve. For more information about the index values, see awvGetWaveNameList.<br><br>**Note:** It is mandatory to input a waveform index value, otherwise, an error message is displayed. |
| *t_parentGroup* | Name of the parent group when the bookmark is organized into a group or nested groups. |
| *t_description* | Description to be added on the bookmark. |
| *g_visible* | Boolean to indicate whether the bookmark is visible by default. Default value: `t` |

*x_subwindow* Subwindow ID within the specified window in which the bookmark is to be created.

*l_proplist* List of properties that can be set on a bookmark. For example:

- `fillColor`—Default value: `blue`
- `font`
- `lineStyle`—`solid`, `dash`, `dot`.
  Default value: `solid` for region bookmark and `dash` for xrange and yrange bookmarks
- `foreground`—Default value: `lightblue`
- `notation`—`suffix`, `scientific`, `engineering`
  Default value: `suffix`
- `showLabel`—`on`, `off`, `on when hover`.
  Default value: `on when hover`
- `sigDigitsMode`—`Auto`, `Manual`.
  Default value: `Auto`
- `significantDigits`—Default value: `4`
- `zoomScaleFactor`—Default value: `1.5`
- `defaultLabel`—Adds a string to the default label of the bookmark.

Some Examples:

- ```
  awvPlaceBookmark(window "region"
  list(100.0ns 0 300.0ns 3) ?waveIndex 1
  ?description "bookmark_1 " ?visible t
  ?subwindow 1 ?properties list(
  list("fillColor" "yellow")
  list("sigDigitsMode" "Manual")
  list("significantDigits" "2")))
  ```
- ```
  awvPlaceBookmark( currentWindow() "xrange"
  list(50ns 150ns) ?waveIndex 1 ?description
  "MyBookMark" ?properties
  list(list("fillColor" "red")
  list("defaultLabel" "defaultBookMark")))
  ```

### Values Returned

| | |
|---|---|
| *l_bookmarkID* | Returns a identification number for the bookmark, which is displayed in the form of a list. |
| `nil` | Returns `nil` if there is an error. |

### Example

The below example creates a bookmark `BM1` of type `xrange` with the description `MyBookMark` in the current window. The bookmark is applied to the specified range of X-axis values: `50—150ns`.

A string `defaultBookMark` is also added to the default label of the bookmark `BM1`.

```
bm=awvPlaceBookmark( currentWindow() "xrange" list(50ns 150ns) ?waveIndex 1
?description "MyBookMark" ?properties list(list("fillColor" "red")
list("defaultLabel" "defaultBookMark")))
=> ("xrangeBookmark[1.1.1]")
```

## awvPlaceWaveformLabel

```
awvPlaceWaveformLabel(
     w_windowId
     x_waveIndex
     l_location
     t_label
     t_expr
     [ ?textOffset  g_textOffset ]
     [ ?color       t_color ]
     [ ?justify     t_justify ]
     [ ?fontStyle   t_fontStyle ]
     [ ?height      t_height ]
     [ ?orient      t_orient ]
     [ ?subwindow   x_subwindow ]
     )
     => s_labelId | nil
```

### Description

Attaches a label to the specified waveform curve in a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *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. |
| *t_expr* | String containing an expression that is evaluated once when the command is issued, and re-evaluated at the completion of each simulation in the auto-update mode. *t_expr* can be `nil`. |
| `?textOffset` *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: `t` |
| `?color` *t_color* | Color for the waveform label. The colors that are available are defined in your technology file. Valid Values: `y1` through `y66` |
| `?justify` *t_justify* | Justification for the label text. Valid Values: `lowerLeft, centerLeft, upperLeft, lowerCenter, centerCenter, upperCenter, lowerRight, centerRight, upperRight` |
| `?fontStyle` *t_fontStyle* | Font style for the label text. Valid Values: `stick, fixed, euroStyle, gothic, math, roman, script, swedish, milSpec` |
| `?height` *t_height* | Height for the label. Valid Values: `small, medium, large` |
| `?orient` *t_orient* | Orientation for the label. Valid Values: `R0`, which specifies horizontal display, and `R90`, which specifies vertical display. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| *s_labelId* | Returns an identification number for the waveform label. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvPlaceWaveformLabel( window(2) 1 list(0 3.5) "Input" nil )
```

Attaches the label *Input* to waveform curve 1 at the location (0 3.5).

### 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).

**Case 1:**

```
awvPlaceWaveformLabel(awvGetCurrentWindow() 1 list( -0.5 0.5 ) "Label1" nil
?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:**

```
awvPlaceWaveformLabel(awvGetCurrentWindow() 1 list( 80MHz -0.5 ) "MARKER " nil)
```

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).
```

### Related Functions

To display a label in a subwindow, see the awvPlaceWindowLabel function on page-127.

To remove the label, or all the labels identified in a list, from a subwindow, see the awvRemoveLabel function on page-165.

## awvPlaceWindowLabel

```
awvPlaceWindowLabel(
     w_windowId
     l_location
     t_label
     t_expr
     [ ?color       t_color ]
     [ ?justify     t_justify ]
     [ ?fontStyle   t_fontStyle ]
     [ ?height      t_height ]
     [ ?orient      t_orient ]
     [ ?subwindow   x_subwindow ]
     )
     => s_labelId | nil
```

### Description

Displays a label in a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *l_location* | List of two Waveform window coordinates. |
| *t_label* | Label for the subwindow. |
| *t_expr* | String containing an expression that is evaluated once when the command is issued and re-evaluated at the completion of each simulation in the auto-update mode. *t_expr* can be `nil`. |
| `?color` *t_color* | Color for the waveform label. The colors that are available are defined in your technology file. Valid Values: `y1` through `y66` |
| `?justify` *t_justify* | Justification for the label text. Valid Values: `lowerLeft`, `centerLeft`, `upperLeft`, `lowerCenter`, `centerCenter`, `upperCenter`, `lowerRight`, `centerRight`, `upperRight` |
| `?fontStyle` *t_fontStyle* | Font style for the label text. Valid Values: `stick`, `fixed`, `euroStyle`, `gothic`, `math`, `roman`, `script`, `swedish`, `milSpec` |
| `?height` *t_height* | Height for the label. Valid Values: `small`, `medium`, `large` |
| `?orient` *t_orient* | Orientation for the label. Valid Values: `R0`, which specifies horizontal display, and `R90`, which specifies vertical display. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| *s_labelId* | Returns an identification number for the subwindow label. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvPlaceWindowLabel( window(2) list(0 0.5) "R5 = " "varR5")
```

If the value of `varR5` is `1K`, the label `R5 =1K` is displayed at the Waveform window location (0 0.5).

### Related Functions

To remove the label, or all the labels identified in a list, from a subwindow, see the awvRemoveLabel function on page-165.

To attach a label to a particular waveform curve in a subwindow, see the awvPlaceWaveformLabel function on page-124.

## awvPlaceXMarker

```
awvPlaceXMarker(
     w_windowId
     n_xLoc
     [ ?label t_label ]
     [ ?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. |
| ?label *t_label* | Marker label you want to set. |
| ?subwindow *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. |

### Examples

```
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
     [ ?label t_label ]
     [ ?subwindow x_subwindowId ]
     [ ?stripNum x_stripNumber ]
     )
     => 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. |
| ?label *t_label* | Marker label you want to set. |
| ?subwindow *x_subwindowId* | Waveform subwindow ID. |
| ?stripNum *x_stripNumber* | Strip number to identify the strip in which the marker is to be placed. |

### 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. |

### Examples

- Paces a horizontal marker with a label, myHorizontalMarker1, at Y= 2.0 in the strip number 1 of the current subwindow of the current graph window:

  ```
  awvPlaceYMarker(gw 2.0 ?label "myHorizontalMarker1" ?subwindow gsw ?stripNum
  1)
  ```

  Where, gw= awvGetCurrentWindow() and gsw= awvGetCurrentSubwindow(gw)

- Paces a horizontal marker with a label, myHorizontalMarker2, at Y= 2.0 in the strip number 2 of the current subwindow of the current graph window:

  ```
  awvPlaceYMarker(gw 2.0 ?label "myHorizontalMarker2" ?subwindow gsw ?stripNum
  2)
  ```

  Where, gw= awvGetCurrentWindow() and gsw= awvGetCurrentSubwindow(gw)

## awvPlotExpression

```
awvPlotExpression(
     w_windowId
     t_expr
     l_context
     [ ?expr   l_dispExprList ]
     [ ?index  l_waveIndexList ]
     [ ?color  l_colorList ]
     [ ?lineType    l_styleList ]
     [ ?dataSymbol  l_symbolList ]
     [ ?subwindow   x_subwindow ]
     [ ?yNumber     l_yNumberList ]
     [ ?stripNumber l_stripNumberList ]
     [ ?showSymbols l_showList ]
     [ ?lineStyle   l_styleList ]
     [ ?lineThickness l_thicknessList ]
     )
     => t / nil
```

### Description

Evaluates the *t_expr* expression and assigns the numbers specified in *l_waveIndexList* to the waveforms resulting from the evaluation.

Any existing waveforms with these numbers are overwritten. If you do not specify *l_waveIndexList*, the lowest numbers for the window are assigned. You can provide an optional list of strings, *l_dispExprList*, with this function call. When you supply this list, the strings are displayed in the Waveform window instead of the expressions.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *t_expr* | String containing an expression that is evaluated once when the command is issued and reevaluated at the completion of each simulation in the auto-update mode. *t_expr* can be nil. |
| *l_context* | Data context for a particular simulation. If evaluating the expressions requires data generated during a simulation, you must specify this argument. Otherwise, specify `nil`. |
| `?expr` *l_dispExprList* | List of strings to display in the Waveform window instead of the expressions. |
| `?index` *l_waveIndexList* | List of integer identifiers to assign to the waveform curves. |
| `?color` *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`" |
| `?lineType` *l_styleList* | List specifying the line styles for the waveforms. If you do not supply this argument, the default line style is used. Valid Values: `line`, `bar`, `scalarPlot` and `polezero`. Default Value: `Line`. |
| `?dataSymbol` *l_symbolList* | List of symbols to use if you want the system to plot data points only (that is, not curves). The system places a symbol on each data point and does not use a line to connect the points. To use a symbol, specify an integer or a single character corresponding to the symbol, as listed in Symbol List. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |
| `?yNumber` *l_yNumberList* | List of numbers identifying the Y axes. |
| `?stripNumber` *l_stripNumberList* | List of numbers identifying the strips. |

| | |
|---|---|
| `?showSymbols` *`l_showList`* | Flag or list of flags that specifies if the symbols for the traces should be plotted or not. By default, the value is set as `nil` and none of the symbols is plotted. Set the value as `t` to show symbols for all the traces. To set the show status for each symbol, specify a list of flags, one flag for each symbol. The number of flags should match the number of symbols in `l_symbolList`. Each flag specifies if the corresponding symbol in the `l_symbolList` list will be shown or not. Default Value: `nil` |
| `?lineStyle` *`l_styleList`* | Specifies the line-style of the signal. Valid values: `Solid`, `Dotted`, `Dashed`, `Dotdashed` |
| `?lineThickness` *`l_thicknessList`* | Specifies the thickness of the signal. Valid values: `Fine`, `Medium`, `Bold` |

Symbol List

| To use the symbol | Enter integers | Enter character |
|---|---|---|
| + (plus) | 0, 10 or 20 | + |
| . (dot) | 1 | . |
| x | 2, 9, 19 | x or X |
| square | 3, 5, 15 | Not supported |
| circle | 4, 6 or 16 | o or O |
| box | 5 | Not supported |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when *`t_expr`* expression is evaluated and the resulting waveforms are plotted. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvPlotExpression( window( 2 )
    "expr( x sin( x ) linRg( 0 1.5 .5 ) )" nil
    ?expr list( "sine" ) ?index list( 3 ) ?color list( "y2" ) )
```

Displays `sin(x)` from $x = 0.0$ to $x = 1.5$. The expression is evaluated at $x = 0.0$, 0.5, 1.0 and 1.5. The waveform for `sin(x)` is assigned number 3. The waveform is labeled `sine` in the Waveform window, and `sin(x)` is displayed in the *y2* layer color.

## awvPrintWaveform

```
awvPrintWaveform(
     [ ?output t_filename | p_port ]
     [ ?numSigDigits x_sigDigits ]
     [ ?format      s_format ]
     [ ?numSpaces   x_numSpaces ]
     [ ?width       x_width ]
     [ ?from        x_from ]
     [ ?to          x_to ]
     [ ?step        x_step ]
     o_waveform1   [o_waveform2 ...]
     )
     => t / nil
```

### Description

Prints the text data of the waveforms specified in the list of waveforms in a result display window.

If you provide a filename as the `?output` argument, this function opens the file and writes the information to it. If you provide a port (the return value of the SKILL `outfile` function), the `awvPrintWaveform` function appends the information to the file that is represented by the port. If you do not provide any argument, it opens a Result Display window and displays the data there. There is a limitation of `awvPrintWaveform` function for precision. It works upto 30 digits for the Solaris port and 18 digits for HP and AIX. If the data is too lengthy to be displayed in the print window, a pop-up form appears indicating this and notifying you that the data will be sent to a default output file or to a filename you specify.

### Arguments

| | |
|---|---|
| *t_filename* | File in which to write the information. The function opens the file, writes to it, and closes it. |
| *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` function for more information. |
| `?numSigDigits` *x_sigDigits* | The number of significant digits to print. This value overrides any global precision value set with the `setup` command.<br>Valid values: `1` through `16` Default value: `6` |
| `?format` *s_format* | The format represents the notation for printed information. This value overrides any global format value set with the `setup` command.<br>Valid values: `'suffix`, `'engineering`, `'scientific`, `'none`<br>Default value: `'suffix`<br>The format for each value is `'suffix`: `1m`, `1u`, `1n`,... `'engineering`: `1e-3`, `1e-6`, `1e-9`, ...; `'scientific`: `1.0e-2`, `1.768e-5`, ...;`'none`.<br>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. |
| `?numSpaces` *x_numSpaces* | The number of spaces between columns. Valid values: `1` or greater Default value: `4` |
| `?width` *x_width* | The width of each column. Valid values: `4` or greater Default value: `14` |
| `?from` *x_from* | The start value at x axis for the waveform to be printed. |
| `?to` *x_to* | The end value at x axis for the waveform to be printed. |
| `?step` *?step* | The step by which text data to be printed is incremented. |
| *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. |

### Examples

```
awvPrintWaveform( v( "/net56" ) )
=> t
```

Prints the text for the waveform for the voltage of `net56`.

```
awvPrintWaveform( 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`.

```
awvPrintWaveform( ?output "myFile" v( "net55" ) )
=> t
```

Prints the text for the specified waveform to a file named `myFile`.

```
awvPrintWaveform( ?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`.

### Related Functions

To plot the Y values in a list against the X values in a list and display the resulting waveforms in a subwindow, see the awvPlotList function on page-141.

To plot the waveforms in a list in a subwindow, see the awvPlotWaveform function on page-151.

To evaluate an expression and specify whether to add the resulting waveforms to the set of existing waveforms, or overwrite the existing waveforms, see the awvSimplePlotExpression function on page-209.

To evaluate an expression and add the resulting waveforms to a subwindow, see the awvAppendExpression function on page-35.

To plot the Y values in a list against the X values in a list and add the resulting waveforms to a subwindow, see the awvAppendList function on page-38.

To add the waveforms in a list to a subwindow, see the awvAppendWaveform function on page-41.

## awvPlotList

```
awvPlotList(
     w_windowId
     l_YListList
     l_XList
     [ ?expr        l_exprList ]
     [ ?index       l_waveIndexList ]
     [ ?color       l_colorList ]
     [ ?lineType    l_typeList ]
     [ ?lineStyle   l_styleList ]
     [ ?lineThickness l_thicknessList ]
     [ ?showSymbols l_showList ]
     [ ?dataSymbol  l_symbolList ]
     [ ?barBase     t_barBase ]
     [ ?barWidth    t_barWidth ]
     [ ?barShift    t_barShift ]
     [ ?subwindow   x_subwindow ]
     [ ?yNumber     l_yNumberList ]
     [ ?stripNumber l_stripNumberList ]
     )
     => t / nil
```

### Description

Plots the Y-axis values in *l_YListList* against the X-axis values in *l_XList* and displays the resulting waveforms in a subwindow. If you do not specify numbers for the waveforms in *l_waveIndexList*, the lowest unused numbers for the subwindow are assigned.

### Arguments

*w_windowId* Waveform Window ID.

*l_YListList* List of lists that specifies different Y values. Each list must have the same number of elements.

*l_XList* List that specifies different X values. The number of elements in this list must equal the number of elements in each list in *l_YListList*

?expr *l_exprList* Specifies the expressions to display by the waveform identifiers. If you do not supply this argument, no expressions are displayed beside the waveform identifiers.

?index *l_waveIndexList*

List of integers identifiers for waveform curves.

?color *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"

?lineType *l_typeList*

List specifying the line type you want to represent the trace with. If you do not supply this argument, the default line type is used.

Valid Values: line, bar, scatterPlot, and polezero

Default Value: line.

?lineStyle *l_styleList*

List specifying the trace styles for the waveform. If you do not supply this argument, the default line style is used.

Valid Values: solid, dash, dot, dashdot and dashdotdot. Default Value: solid.

?lineThickness *l_thicknessList*

Specifies the thickness or width of the trace.

Valid Values: fine, medium, thick, and extrathick.

Default Value: fine.

?showSymbols *l_showList*

Flag or list of flags that specifies if the symbols for the traces should be plotted or not. By default, the value is set to `nil` and none of the symbols are plotted. Set the value to `t` to show symbols for all the traces. To set the show status for each symbol, specify a list of flags, one flag for each symbol. The number of flags should match the number of symbols in `l_symbolList`. Each flag specifies if the corresponding symbol in the `l_symbolList` list will be shown or not.

`?dataSymbol` *`l_symbolList`*

List of symbols to use if you want the system to plot data points only (that is, not curves). The system places a symbol on each data point and does not use a line to connect the points. To use a symbol, specify an integer or a single character corresponding to the symbol, as listed in Symbol List.

`?subwindow` *`x_subwindow`*

Number for the subwindow (found in the upper right corner). Default Value: Current subwindow

`?yNumber` *`l_yNumberList`*

List of numbers identifying the Y axes.

`?stripNumber` *`l_stripNumberList`*

List of numbers identifying the strips.

Symbol List

| To use the symbol | Enter integers | Enter character |
|---|---|---|
| + (plus) | 0, 10 or 20 | + |
| . (dot) | 1 | . |
| x | 2, 9, 19 | x or X |
| square | 3, 5, 15 | Not supported |
| circle | 4, 6 or 16 | o or O |
| box | 5 | Not supported |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the specified Y values are plotted against the specified X values and the waveforms are displayed in the subwindow. |
| `nil` | Returns `nil` if there is an error. |

### Example

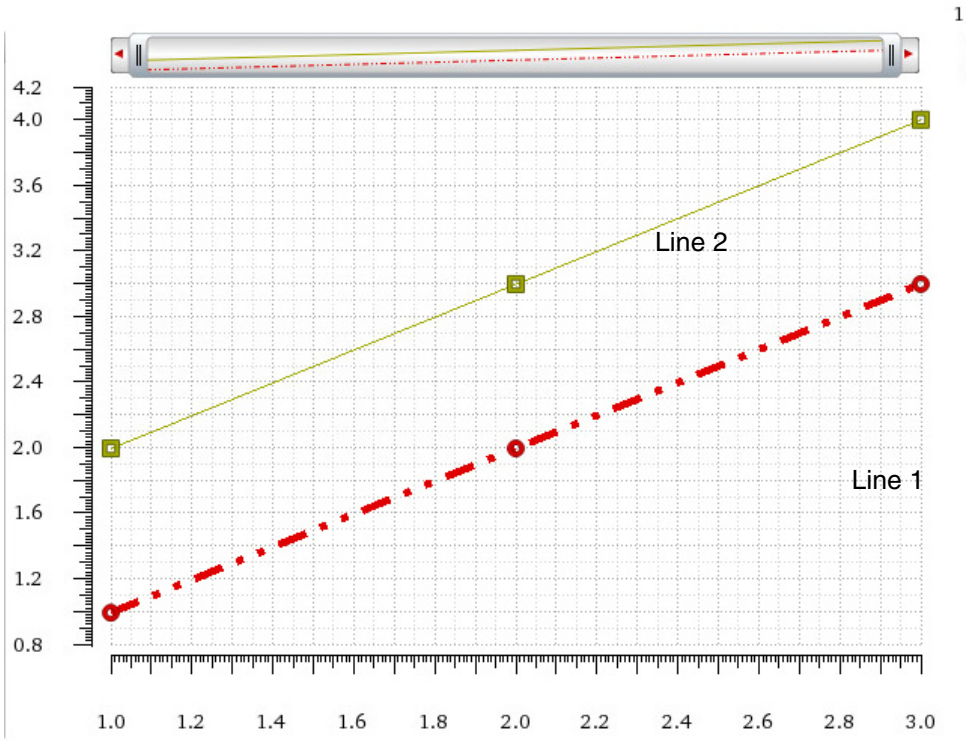
Open the Virtuoso Visualization and Analysis XL window using the following function:

```
w=awvCreatePlotWindow()
```

Run the following function in CIW:

```
awvPlotList(w list( list( 1 2 3 ) list( 2 3 4 ) ) list( 1 2 3 ) ?index list( 1 3 )
?lineType list("line" "line") ?lineStyle list("dashdotdot" "solid") ?lineThickness
list("extrathick" "fine") ?showSymbols list(t t) ?dataSymbol list(4 5))
```

The function returns two straight lines by plotting the Y-axis values (`1`, `2`, `3`) and (`2`, `3`, `4`) against the X-axis values (`1`, `2`, `3`), as shown in the following figure:




| | **Line 1** | **Line 2** |
|---|---|---|
| `lineType` | line | line |
| `lineStyle` | dashdotdot | solid |
| `lineThickness` | extrathick | fine |
| `dataSymbol` | circle | square |

### Related Functions

To evaluate an expression and assign the numbers specified in a list to the waveforms resulting from the evaluation, see the awvPlotExpression function on page-133.

To plot the waveforms in a list in a subwindow, see the awvPlotWaveform function on page-151.

To evaluate an expression and specify whether to add the resulting waveforms to the set of existing waveforms, or overwrite the existing waveforms, see the awvSimplePlotExpression function on page-209.

To evaluate an expression and add the resulting waveforms to a subwindow, see the awvAppendExpression function on page-35.

To plot the Y values in a list against the X values in a list and append the resulting waveforms to a subwindow, see the awvAppendList function on page-38.

To add the waveforms in a list to a subwindow, see the awvAppendWaveform function on page-41.

## awvPlotSignals

```
awvPlotSignals(
     l_sigList
     [ ?plotStyle t_style ]
     [ ?graphType t_type ]
     [ ?graphModifier l_modifier ]
     [ ?wavetype g_waveType ]
     )
     => o_waveform / nil
```

### Description

Displays a signal in the graph window.

### Arguments

| | |
|---|---|
| *l_sigList* | List of signals to be plotted specified in the following format: (*list* (*list resultsDir1* (*list* (*list result1* (*list signal1 signal2* ...) ...) ...))) Remember to put quotation marks before and after each signal name. |
| ?plotStyle *t_plotStyle* | Plot destination. Valid values: Append, Replace, New Window, New Subwindow<br>Remember to put quotation marks before and after the style. |
| ?graphType *t_graphType* | Type of plot. Valid values: Default, Rectangular, Histogram, Polar, Impedance Admittance, RealvsImag<br>Remember to put quotation marks before and after the graph type. |
| ?graphModifier *t_graphModifier* | X axis of graph for rectangular graphs Valid values: Magnitude, Phase, WPhase, Real, Imaginary, dB10, dB20 |
| ?waveType *g_waveType* | Specifies whether the signal is Y versus Y or not. Valid values: YvsY, nil |

### Value Returned

| | |
|---|---|
| *o_waveform* | Returns t when the signal is plotted. |
| nil | Returns nil if there is an error. |

## Example

```
awvPlotSignals(`(("./ampsim.raw" (("ac-ac" ("net10"))))) ?plotStyle "Append"
?graphType "Default")
```

## awvPlotSimpleExpression

```
awvPlotSimpleExpression(
     t_expression
     [ ?plotStyle t_plotStyle ]
     [ ?graphType t_graphType ]
     [ ?graphModifier t_graphModifier ]
     )
     => o_waveform / nil
```

### Description

Evaluates an expression and plots the resulting waveform.

### Arguments

| | |
|---|---|
| *t_expression* | Expression to be evaluated and plotted. |
| ?plotStyle *t_plotStyle* | Plot destination.<br>Valid values: `Append`, `Replace`, `New Window`, `New Subwindow`<br>Remember to put quotation marks before and after the style. |
| ?graphType *t_graphType* | Type of plot.<br>Valid values: `Default`, `Rectangular`, `Histogram`, `Polar`, `Impedance Admittance`, `RealvsImag`<br>Remember to put quotation marks before and after the graph type. |
| ?graphModifier *t_graphModifier* | X-axis of graph for rectangular graphs<br>Valid values: `Magnitude`, `Phase`, `WPhase`, `Real`, `Imaginary`, `dB10`, `dB20` |

### Examples

```
awvPlotSimpleExpression("v(\"/net6\" ?result \"tran-tran\") - v(\"/out\" ?result
\"tran-tran\")" ?plotStyle "Append" ?graphType "Default" ?graphModifier
"Magnitude")
```

```
awvPlotSimpleExpression("leafValue( getData(\"top.b[0:7]\" ?resultsDir \"./
simulation/amslib/top/adexl/results/data/Interactive.0/psf/amslib:top:1/psf\"
?result \"tran\") \"idc\" 1e-05 \"temperature\" 34 )" ?plotStyle "New Subwindow"
?graphType "" ?graphModifier "Magnitude")
```

```
awvPlotSimpleExpression("dnl(v(\"13\" ?result \"tran\") v(\"16\" ?result \"tran\")
?mode \"auto\"  ?crossType \"rising\" ?delay 0.0 ?method \"end\" ?units \"abs\"
?nbsamples nil)" ?plotStyle "New Window" ?graphType "" ?graphModifier "Magnitude")
```

```
awvPlotSimpleExpression("dnl(v(\"13\" ?result \"tran\") v(\"16\" ?result \"tran\")
?mode \"auto\"  ?crossType \"rising\" ?delay 0.0 ?method \"end\" ?units \"abs\"
?nbsamples nil)" ?plotStyle "New Window" ?graphType "" ?graphModifier "Magnitude")
awvPlotSimpleExpression("vtime('tran \"/net12\")" ?plotStyle "New Window"
?graphType "" ?graphModifier "Magnitude"))
```

## awvPlotWaveform

```
awvPlotWaveform(
     w_windowId
     l_waveform
     [ ?subwindow   x_subwindow ]
     [ ?yNumber     l_yNumberList ]
     [ ?stripNumber l_stripNumberList ]
     [ ?expr        l_exprList ]
     [ ?index       l_waveIndexList ]
     [ ?component   t_component ]
     [ ?color       l_colorList ]
     [ ?lineType    l_typeList ]
     [ ?lineStyle   l_styleList ]
     [ ?lineThickness l_thicknessList ]
     [ ?dataSymbol  l_symbolList ]
     [ ?showSymbols l_showList ]
     [ ?barBase     t_barBase ]
     [ ?barWidth    t_barWidth ]
     [ ?barShift    t_barShift ]
     [ ?yAxisUnit   t_yAxisUnit ]
     [ ?yAxisLabel  t_yAxisLabel ]
     [ ?label       t_label ]
     [ ?modifierType     t_modifierType ]
     [ ?graphType   t_graphType ]
     )
     => t / nil
```

### Description

Plots the waveforms in the *l_waveform* list in a subwindow. If you do not specify numbers for the waveforms in *l_waveIndexList*, the lowest *unused* numbers for the subwindow are assigned.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *l_waveform* | List of waveform objects. |
| ?expr *l_exprList* | Specifies the expressions to display next to the waveform identifiers. If you do not specify *l_exprList*, no expressions are displayed beside the waveform identifiers. |
| ?index *l_waveIndexList* | List of integer identifiers to assign to the waveform curves. |
| *t_component* | Plot option for the waveform. This option is ignored unless the display mode is set to *strip* or *composite*. (The imaginary part of the waveform is plotted against its real part when the display mode is Smith.)<br>Default value: real<br>Valid values: See Component List. |
| ?color *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. The color information can also be provided in the RGB format, such as `0X00FF50`.Valid Values: "`y1`" through "`y66`" |
| ?lineType *l_typeList* | List specifying the line styles for the waveforms. If you do not supply this argument, the default line style is used.<br>Valid Values: `line`, `bar`, `scatterPlot` and `polezero`.<br>Default Value: `line`. |
| ?lineStyle *l_styleList* | |
| ?dataSymbol *l_symbolList* | List of symbols to use if you want the system to plot data points only (that is, not curves). The system places a symbol on each data point and does not use a line to connect the points. To use a symbol, specify an integer or a single character corresponding to the symbol, as listed in Symbol List. |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner).<br>Default Value: Current subwindow |

| | |
|---|---|
| `?yNumber` *`l_yNumberList`* | List of numbers identifying the Y axes. |
| `?stripNumber` *`l_stripNumberList`* | List of numbers identifying the strips. |
| `?showSymbols` *`l_showList`* | Flag or list of flags that specifies if the symbols for the traces should be plotted or not. By default, the value is set as `nil` and none of the symbols are plotted. Set the value as `t` to show symbols for all the traces. To set the show status for each symbol, specify a list of flags, one flag for each symbol. The number of flags should match the number of symbols in `l_symbolList`. Each flag specifies if the corresponding symbol in the `l_symbolList` list will be shown or not.<br>Default Value: nil |
| `?lineStyle` *`l_styleList`* | Specifies the line-style of the signal. Valid values: `Solid`, `Dotted`, `Dashed`, `Dotdashed` |
| `?lineThickness` *`l_thicknessList`* | Specifies the thickness of the signal. Valid values: `Fine`, `Medium`, `Bold` |
| `?barBase` *`t_barBase`* | |
| `?barWidth` *`t_barWidth`* | Specifies the width of the bar when *`lineType`* is set as `bar`.<br>Valid Values: Any integer value<br>Default value: `1` |
| `?barShift` *`t_barShift`* | Specifies whether to shift the bar ahead or backwards when *`lineType`* is set as `bar`.<br>Valid Values: Positive integer for backward shift and negative integer for forward shift<br>Default value: `0` |
| `?yAxisUnit` *`t_YAxisUnit`* | Specifies the unit for Y-axis.<br>Valid Values: Any string value<br>Default value: blank (nil) |
| `?yAxisLabel` *`t_yAxisLabel`* | Specifies the Y-axis label.<br>Valid Values: Any string value<br>Default value: blank (nil) |
| `?label` *`t_label`* | Displays the label of the waveform that is displayed in Name column of trace legend.<br>Valid Values: Any string value<br>Default value: Name of the waveform expression. |

| | |
|---|---|
| `?modifierType`<br>*`t_modifierType`* | |
| `?graphType`<br>*`t_graphType`* | Specifies the type of the graph.<br>Valid Values: `Rectangular`, `polar`, `impedance`, `admittance`<br>Default value: `Rectangular` |

Symbol List

| To use the symbol | Enter integers | Enter character |
|---|---|---|
| + (plus) | 0, 10 or 20 | + |
| . (dot) | 1 | . |
| x | 2, 9, 19 | x or X |
| square | 3, 5, 15 | Not supported |
| circle | 4, 6 or 16 | o or O |
| box | 5 | Not supported |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the waveforms specified in the *`l_waveform`* list are plotted. |
| `nil` | Returns `nil` if there is an error. |

### Examples

```
awvPlotWaveform( window( 2 ) list( w1 w2 ) ?expr
    list( "wave1" "wave2" ) )
```

Plots waveforms *w1* and *w2* in the Waveform window and assigns them numbers 1 and 2, respectively. The *wave1* and *wave2* expressions are displayed next to their waveform identifiers.

Consider the following two waveforms:

```
wave1 = drCreateWaveform( drCreateVec('double list(1 2 3 4 5)) drCreateVec('double
list(10 20 30 40 50)))
wave2 = drCreateWaveform( drCreateVec('double list(1 2 3 4 5)) drCreateVec('double
list(11 21 31 41 51)))
```

Open the Virtuoso Visualization and Analysis XL window using the following function:

```
win=awvCreatePlotWindow()
```

Below are some examples showing how different arguments of this function work:

- Plots `wave2` and `wave1` with the colors `y6` and `y2` from the color bank.

  ```
  awvPlotWaveform(currentWindow() list(wave2 wave1) ?color list("y6" "y2"))
  ```

- Plots `wave2` with the style bar and *`barWidth`* set to `20`:

  ```
  awvPlotWaveform(currentWindow() list(wave2) ?lineType list("bar") ?barWidth
  list(20))
  ```

- Plots with dotted lines:

  ```
  awvPlotWaveform(currentWindow() list(wave2) ?lineStyle list("dotted"))
  ```

- Plots in bold:

  ```
  awvPlotWaveform(currentWindow() list(wave2) ?lineThickness list("bold"))
  ```

- Shifts the bar graph:

  ```
  awvPlotWaveform(currentWindow() list(wave2) ?lineType list("bar") ?barShift
  list(50))
  ```

- Adds a name to the plot:

  ```
  awvPlotWaveform(currentWindow() list(wave2) ?label list("myGraph"))
  ```

- Plots with the graph type admittance:

  ```
  awvPlotWaveform(currentWindow() list(wave2) ?graphType list("admittance"))
  ```

- Adds another Y-axis:

  ```
  awvPlotWaveform(currentWindow() list(wave2) ?yNumber list(2))
  ```

- Plots on another strip:

  ```
  awvPlotWaveform(currentWindow() list(wave2) ?stripNumber list(3))
  ```

- Plots data points on waveform:

  ```
  awvPlotWaveform(awvGetCurrentWindow() list(wave2) ?showSymbols t ?dataSymbol
  list("o"))
  ```

- Sets the Y-axis units and label:

  ```
  awvPlotWaveform(currentWindow() list(wave2) ?yaxisUnit list("V"))
  awvPlotWaveform(currentWindow() list(wave2) ?yaxisLabel list("volts"))
  ```

- Plots two waves `wave1` and `wave2` with indexes as `1` and `2` respectively:

  ```
  awvPlotWaveform(awvGetCurrentWindow() list(wave1 wave2) ?index list(1 2))
  ```

Replaces the wave with index=2 from the graph, which means `wave2` is replaced by `wave1`:

```
awvPlotWaveform(awvGetCurrentWindow() list(wave1) ?index list(2))
```

### Related Functions

To delete all the waveforms in a subwindow, see the awvDeleteAllWaveforms function on page-58.

To evaluate an expression and assign the numbers specified in a list to the waveforms resulting from the evaluation, see the awvPlotExpression function on page-133.

To plot the Y values in a list against the X values in a list and display the resulting waveforms in a subwindow, see the awvPlotList function on page-141.

To evaluate an expression and specify whether to add the resulting waveforms to the set of existing waveforms, or overwrite the existing waveforms, see the awvSimplePlotExpression function on page-209.

To evaluate an expression and add the resulting waveforms to a subwindow, see the awvAppendExpression function on page-35.

To plot the Y values in a list against the X values in a list and add the resulting waveforms to a subwindow, see the awvAppendList function on page-38.

To add the waveforms in a list to a subwindow, see the awvAppendWaveform function on page-41.

## awvPlotWaveformOption

```
awvPlotWaveformOption(
     w_windowId
     x_waveIndex
     t_component
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Sets the plot option for a particular waveform in a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform Window ID. |
| *x_waveIndex* | Integer identifying the waveform curve. |
| *t_component* | Plot option for the waveform. This option is ignored unless the display mode is set to *strip* or *composite*. (The imaginary part of the waveform is plotted against its real part when the display mode is Smith.) Valid values are shown in the table below. |
| ?subwindow<br>*x_subwindow* | Number for the subwindow (found in the upper right corner).<br>Default Value: Current subwindow |

### *Component List*

| | |
|---|---|
| magnitude | Plots the magnitude of the waveform against an independent variable. |
| dB10 | Calculates *10*log* of the magnitude of each point in the waveform and plots the results against an independent variable. |
| dB20 | Calculates *20*log* of the magnitude of each point in the waveform and plots the results against an independent variable. |
| dBm | Adds 30 to the value calculated by dB10. |
| phaseDeg | Plots the phase, in degrees, of the waveform against an independent variable. |
| phaseRad | Plots the phase, in radians, of the waveform against an independent variable. |
| wrappedPhaseDeg | Plots the wrapped phase, in degrees, of the waveform against an independent variable. |
| wrappedPhaseRad | Plots the wrapped phase, in radians, of the waveform against an independent variable. |
| real | Plots the real part of the waveform against an independent variable. |

| | |
|---|---|
| `imaginary` | Plots the imaginary part of the waveform against an independent variable. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the plot option is set for the waveform. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvPlotWaveformOption( window( 2 ) 1 "phaseDeg" )
```

Displays the phase (in degrees) of the waveform whose index is one.

## awvRedisplaySubwindow

```
awvRedisplaySubwindow(
     w_windowId
     [ ?subwindow   x_subwindow ]
     g_readData
     )
     => t / nil
```

### Description

Refreshes the display for a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |
| *g_readData* | Boolean parameter. When set to `t`, graph reads the simulation data, refreshes the subwindow with new data and updates the traces if necessary. If set to `nil`, a simple UI refresh takes place for the graph subwindow. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the subwindow is redrawn. |
| `nil` | Returns `nil` if the Waveform window or the subwindow do not exist. |

### Example

```
awvRedisplaySubwindow( window(2) ?subwindow 3 )
=> t
```

Refreshes the display for subwindow 3.

### Related Function

To refresh the display for a Waveform window, see the awvRedisplayWindow.

## awvRedrawWindowMenuCB

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

### Description

Redraws all the objects (e.g. waveforms, markers) in the waveform window.

The function is defined in `dfII/etc/context/awv.cxt`.

### Arguments

None.

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the command is successful. |
| `nil` | Otherwise, returns `nil`. |

### Example

```
awvRedrawWindowMenuCB() => t
```

## awvRedisplayWindow

```
awvRedisplayWindow(
     w_windowId
     )
     => t / nil
```

### Description

Refreshes the display for a Waveform window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the Waveform window is redrawn. |
| `nil` | Returns `nil` if the specified Waveform window does not exist. |

### Related Function

To refresh the display for a subwindow, see the awvRedisplaySubwindow function on page-160.

## awvRemoveDate

```
awvRemoveDate(
     w_windowId
     )
     => t / nil
```

### Description

Removes the date and time from the Waveform window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the date and time are removed. |
| `nil` | Returns `nil` if the specified Waveform window does not exist. |

### Related Function

To display the date and time in a Waveform window, see the awvDisplayDate function on page-66.

## awvResumeViVA

```
awvResumeViVA()
```

### Description

This function, when called from CIW, resumes ViVA (if suspended) provided all the licensing requirements are met.

### Arguments

None

### Value Returned

None

### Example

```
awvResumeViVA()
```

## awvRemoveLabel

```
awvRemoveLabel(
     w_windowId
     s_id | l_id
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Removes the label, or all the labels identified in a list, from a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *s_id* | Label to remove. |
| *l_id* | List of labels to remove. |
| ?subwindow<br>*x_subwindow* | Number for the subwindow (found in the upper right corner).<br>Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| t | Returns t when the label or labels are removed. |
| nil | Returns nil if there is an error. |

### Related Functions

To attach a label to a particular waveform curve in a subwindow, see the awvPlaceWaveformLabel function on page-124.

To display a label in a subwindow, see the awvPlaceWindowLabel function on page-127.

## awvRemoveSubwindowTitle

```
awvRemoveSubwindowTitle(
     w_windowId
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Removes the title from a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the title is removed. |
| `nil` | Returns `nil` if there is an error, such as the specified Waveform window does not exist or the specified subwindow does not have a title. |

### Related Functions

To display a title in a subwindow, see the awvDisplaySubwindowTitle function on page-68.

To display a title on a Waveform window, see the awvDisplayTitle function on page-69.

To remove a title from a Waveform window, see the awvRemoveTitle function on page-167.

## awvRemoveTitle

```
awvRemoveTitle(
     w_windowId
     )
     => t / nil
```

### Description

Removes the title from a Waveform window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `nil` when the title is removed. |
| `nil` | Returns `nil` if the specified Waveform window does not exist. |

### Related Functions

To display a title on a Waveform window, see the awvDisplayTitle function on page-69.

To display a title in a subwindow, see the awvDisplaySubwindowTitle function on page-68.

To remove a title from a subwindow, see the awvRemoveSubwindowTitle function on page-166.

## awvResetAllWindows

```
awvResetAllWindows(
     [ ?force  g_force ]
     )
     => t
```

### Description

Resets all the windows returned by `awvGetWindowList`. The contents of the windows are erased, and any subwindows whose update statuses are *on* are deleted. To delete all subwindows, regardless of update status, set *`g_force`* to `t`.

**Note:** The Waveform Windows remain at their current sizes and locations.

### Arguments

| | |
|---|---|
| `?force` *`g_force`* | Boolean flag that specifies whether all the subwindows of the Waveform Windows are deleted, regardless of update status. Valid Values: `t` specifies that all the subwindows are deleted, `nil` specifies that only subwindows whose update statuses are *on* are deleted. Default value: `nil`. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the Waveform Windows are reset. |

### Related Function

To reset a particular Waveform window, see the awvResetWindow function on page-169.

## awvResetWindow

```
awvResetWindow(
     w_windowId
     [ ?force  g_force ]
     )
     => t / nil
```

### Description

Resets a Waveform window to the state of a new window. The contents of the window are erased, and any subwindows whose update statuses are *on* are deleted. To delete all subwindows, regardless of update status, set *g_force* to `t`.

**Note:** The Waveform window remains at its current size and location.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| ?force *g_force* | Boolean flag that specifies whether all the subwindows of the Waveform window are deleted, regardless of update status. Valid Values: `t` specifies that all subwindows are deleted, `nil` specifies that only subwindow whose update statuses are *on* are deleted. Default value: `nil`. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the Waveform window is reset. |
| `nil` | Returns `nil` if the specified Waveform window does not exist. |

### Example

```
awvResetWindow( window(5) ?force t )
=> t
```

Erases the contents of Waveform window 5 and sets the window to the state of a new window. All the subwindows are deleted, regardless of update status.

### Related Function

To reset all the windows returned by `awvGetWindowList`, see the awvResetAllWindows function on page-168.

## awvRfLoadPull

```
awvRfLoadPull(
     w_wave
     [ ?maxValue   x_maxValue ]
     [ ?minValue   x_minValue ]
     [ ?numCont    x_numCont ]
     [ ?closeCont  g_closeCont ]
     [ ?name       t_name ]
     )
     => t / nil
```

### Description

Draws load pull contour for the given waveform of PSS analysis. This function works only for two-dimensional sweep PSS results. The inner sweep should be phase and the outer sweep should be mag.

### Arguments

| | |
|---|---|
| *w_wave* | Signal waveform. |
| ?maxValue *x_maxValue* | Largest value of the contour to be drawn. Default value: `nil`. Specifies that the largest value is to be taken from the results. |
| ?minValue *x_minValue* | Smallest value of the contour to be drawn. Default value: `nil`. Specifies that the smallest value is to be taken from the results. |
| ?numCount *x_numCont* | Number of points on the contour. Default value: `8` |
| ?closeCount *g_closeCont* | Boolean flag that specifies if a closed or open contour is to be drawn. Valid values: `t` specifies that a closed contour is to be drawn. `nil` specifies that an open contours is to be drawn. |
| ?name *t_name* | Name of the contour. Default value: `p` |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the Load Pull contour is drawn. |
| `nil` | Returns `nil` if the specified waveform does not exist. |

### Example

```
awvRfLoadPull(ip3_wave nil nil 9 t "ip3")
=> t
```

## awvSaveWindow

```
awvSaveWindow(
     w_windowId
     t_fileName
     )
     => t / nil
```

### Description

Saves the state of a Waveform window to a file. You can specify a path name, or you can specify only the file name. If you provide only a file name, the file is placed in the directory in which you started the software.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *t_fileName* | Name of the file in which to store the state of the Waveform window. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the window state is stored. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSaveWindow( window(4) "/tmp/my_file" )
=> t
```

Saves the state of Waveform window 4 in *my_file* in the `/tmp` directory.

### Related Function

To initialize a Waveform window from information saved in a file, see the awvLoadWindow function on page-112

## awvSaveWindowImage

```
awvSaveWindowImage(
     w_window
     t_path
     t_filePrefix
     g_cardLayout
     )
     => l_files
```

### Description

Saves the image of a plot window in `.png` format.

**Note:** Support for the `awvSaveWindowImage` function will be removed in a future release. To save the graph as an image, use the `saveGraphImage` function.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *t_fileName* | Path where you want to save the image file. |
| *t_filePrefix* | Prefix string, which is to be prefixed in the name of file being saved. |
| *g_cardLayout* | Card layout. Valid value: `t` if the window is in card layout mode. |

### Value Returned

| | |
|---|---|
| `l_files` | Returns the list of image files saved. |

## awvSaveMenuCB

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

### Description

Displays the *Save (window -> Save...)* Option Menu.

The function is defined in `dfII/etc/context/awv.cxt`.

### Arguments

None

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the command was successful. |
| `nil` | Otherwise, returns `nil`. |

### Example

```
awvSaveMenuCB() => t
```

## awvSaveToCSV

```
awvSaveToCSV(
     l_waveform
     t_fileName
     [ ?from x_from ]
     [ ?to x_to ]
     [ ?precision x_precision ]
     [ ?step x_step ]
     [ ?linLog g_linLog ]
     [ ?exprList l_expressionList ]
     )
     => t / nil
```

### Description

Saves the waveform data to the specified CSV file.

### Arguments

| | |
|---|---|
| *l_waveform* | Input waveform or list of input waveforms. |
| *t_fileName* | Name of the CSV file in which waveform data is to be saved. |
| ?from *x_from* | Starting X-axis value after which the waveform data is to be saved. |
| ?to *x_to* | End X-axis value upto which the waveform data is to be saved. |
| ?precision *x_precision* | Precision value to be used. |
| ?step *x_step* | Step value to be used. |
| ?linLog *g_linLog* | Specifies whether the waveform data is to be saved in a linear or logarithmic format. When this argument is set to true, data is saved in the linear format. When set to false, the logarithmic format is used. |
| ?exprList *l_expressionList* | List of expression names to be printed in the CSV file. |

### Values Returned

| | |
|---|---|
| t | Returns t when function runs successfully. |
| nil | Returns nil if there is an error. |

### Examples

- `awvSaveToCSV(wave filename)`

  This example saves the specified waveform in the given CSV file.

- `awvSaveToCSV( list(wave1 wave2 wave3) filename)`

  This example saves a list of specified waveforms (wave1, wave2, and wave3) in the given CSV file.

- `awvSaveToCSV( wave filename ?from 10n ?to 200n)`

  This function saves the specified waveform in the given CSV file that starts from 10n and ends at 200n.

- `awvSaveToCSV( wave filename ?from 10n ?to 200n ?step 10n)`

- `awvSaveToCSV( wave filename ?from 10n ?to 200n ?step 10n ?linLog "log")`
- `awvSaveToCSV( wave filename ?from 10n ?to 200n ?step 10n ?linLog "log" ?precision 12)`
- Consider the two waveform data objects, `w1` and `w2`, created using:

  ```
  w1 = drCreateWaveform( drCreateVec( 'double list( 1 2 3 4 5))
  drCreateVec( 'double list( 10 20 30 40 50)))
  ```

  ```
  w2 = drCreateWaveform( drCreateVec( 'double list( 1 2 3 4))
  drCreateVec( 'double list( 10 20 30 40)))
  ```

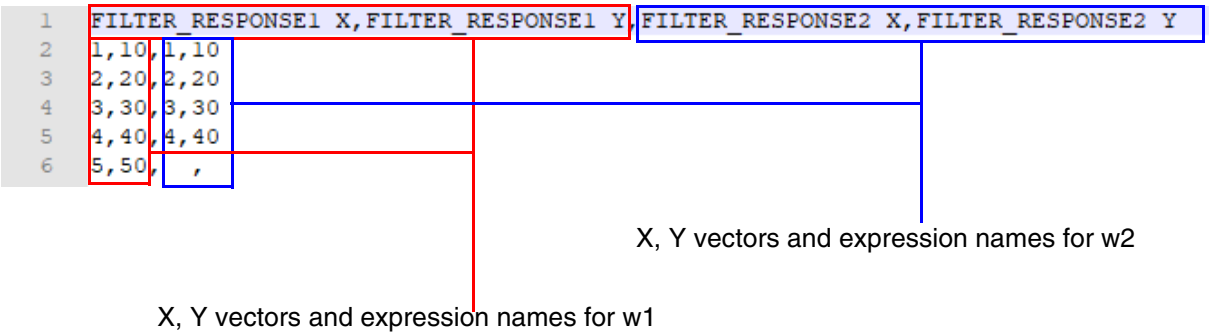
  These waveforms are passed to the following list:

  ```
  wavelist = list(w1 w2)
  ```

  This wavelist is then passed as an argument along with the expression names to be printed in the specified CSV file, `log.csv`, using the `awvSaveToCSV` function.

  ```
  awvSaveToCSV( wavelist "log.csv" ?exprList
  list("FILTER_RESPONSE1" "FILTER_RESPONSE2"))
  ```

  The following figure shows the contents of the `log.csv` file:



## awvSetCurrentSubwindow

```
awvSetCurrentSubwindow(
     w_windowId
     x_subwindow
     )
     => t / nil
```

### Description

Specifies *x_subwindow* as the current subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *x_subwindow* | 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*. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSetCurrentSubwindow( window(2) 1 )
=> t
```

Specifies subwindow 1 as the current subwindow.

### Related Function

To get the current subwindow, see the awvGetCurrentSubwindow function on page-79.

## awvSetCurrentWindow

```
awvSetCurrentWindow(
     w_windowId
     )
     => t / nil
```

### Description

Specifies *w_windowId* as the current Waveform window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID for the window to become the current window. |

### Value Returned

| | |
|---|---|
| t | Returns t when the current window is set. |
| nil | Returns nil if there is an error. |

### Example

```
awvSetCurrentWindow( window(2) )
=> t
```

Specifies window number 2 as the current Waveform window.

### Related Function

To return the window ID for the current Waveform window, see the awvGetCurrentWindow function on page-80.

## awvSetCursorPrompts

```
awvSetCursorPrompts(
     w_windowId
     x_yNumber
     t_xPrompt
     t_yPrompt
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Sets the tracking cursor prompts for the waveforms around a particular Y axis and a particular strip in a subwindow. If you specify `nil` for the prompts, the default prompts are used.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *x_yNumber* | Specifies the Y axis to have the tracking cursor prompts. Valid Values: 1 through 4 |
| *t_xPrompt* | Prompt to put next to the X axis value. |
| *t_yPrompt* | Prompt to put next to the Y axis value. |
| `?stripNumber` *x_stripNumber* | Specifies the strip in which the tracking cursor prompts are to be set. Valid Values: 1 through 20 |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the tracking cursor prompts are set. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSetCursorPrompts( window( 2 ) 1 "Time" "Voltage" )
```

When the tracking cursor is on a waveform on the Y axis (Y1), the tracking cursor banner has the word `Time` next to the X axis value and the word `Voltage` next to the Y axis value.

## awvSetDisplayMode

```
awvSetDisplayMode(
     w_windowId
     t_mode
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Sets the display mode of a subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *t_mode* | String representing the display mode for the subwindow. Valid Values: `strip`, `composite`, or `smith`. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the display mode of the subwindow is set. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSetDisplayMode( window(2) "strip" ?subwindow 3 )
=> t
```

Sets the display of subwindow 3 to strip mode.

### Related Functions

To return the display mode of a subwindow, see the awvGetDisplayMode function on page-81.

## awvSetDisplayStatus

```
awvSetDisplayStatus(
     w_windowId
     g_enable
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Turns the display of a subwindow *on* or *off* based on the value of the `g_enable` flag. You can use this function to hide a subwindow without deleting it.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *g_enable* | Boolean flag that specifies whether the display of a subwindow is turned *on* or *off*. If this argument is set to `nil` and the specified subwindow is displayed, the subwindow display is hidden. If this argument is set to non-`nil` and the specified subwindow is not displayed, the subwindow is displayed again. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the subwindow display is turned *off* or *on*. |
| `nil` | Returns `nil` if the Waveform window or the subwindow do not exist. |

### Example

```
awvSetDisplayStatus( window(2) nil ?subwindow 1 )
=> t
```

Turns *off* the display of subwindow 1.

## awvSetInitializationTimeout

```
awvSetInitializationTimeout(
     x_timeOut
     )
     => x_timeOut
```

### Description

Sets the time-out period (in seconds) for ADE to establish connection with Virtuoso Visualization and Analysis XL and returns the same value. The default value of the time-out period is `120` seconds.

### Arguments

| | |
|---|---|
| *x_timeOut* | Time-out period (in seconds). |

### Value Returned

| | |
|---|---|
| *x_timeOut* | Time-out period (in seconds). |

### Example

```
awvSetInitializationTimeout( 240 )
=> 240
```

This will set the time-out period to `240` seconds, so that ADE will keep trying to establish a connection with Virtuoso Visualization and Analysis XL for up to `240` seconds.

## awvSetLegendWidth

```
awvSetLegendWidth(
     w_windowId
     x_width
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

### Description

Sets the width of the legend in the specified subwindow or window. If no subwindow is specified, the currently selected subwindow is used. This works only when the legend is displayed in the left position.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *x_width* | Width to be set for the legend.<br>Valid values: Any number or `auto`. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner).<br>Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the specified legend width is successfully set. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
openResults("ampsim.raw")
window = awvCreatePlotWindow()
awvPlotWaveform(window list(v("out" ?result "tran-tran")) ?subwindow subwindow)
awvSetLegendWidth(window 250 ?subwindow subwindow)
awvSetLegendWidth(window "auto" ?subwindow subwindow)
```

## awvSetOptionDefault

```
awvSetOptionDefault(
     S_name
     )
     => t / nil
```

### Description

Restores a Waveform window option to its default value. The option takes effect for any Waveform Windows or subwindows that are opened after the option is set.

For a list of Waveform window default options that you can change, see the table at the beginning of this chapter.

### Arguments

| | |
|---|---|
| *S_name* | Name of the option to restore. The option name can be a string or a symbol. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the option is restored to its default value. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSetOptionDefault( "mode" )
=> t
```

Sets the *mode* option back to its default value, which is `composite`.

### Related Functions

To set a Waveform window option to a particular value, see the awvSetOptionValue function on page-187.

## awvSetOptionValue

```
awvSetOptionValue(
     S_name
     g_value
     )
     => g_value | nil
```

### Description

Sets a Waveform window option. The option takes effect for any Waveform Windows or subwindows that are opened after the option is set.

For a list of Waveform window default options that you can change, see the table at the beginning of this chapter.

### Arguments

| | |
|---|---|
| *S_name* | Name of the Waveform window option. The option name can be a string or a symbol. |
| *g_value* | Value for the option. |

### Value Returned

| | |
|---|---|
| *g_value* | Returns the new value of the option. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSetOptionValue( "displayGrids" t )
=> t
```

Turns on the grid display option for a Waveform window.

### Related Functions

To restore a Waveform window option to its default value, see the awvSetOptionDefault function on page-186.

## awvSetOrigin

```
awvSetOrigin(
     w_windowId
     l_origin
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Sets the axis origin of a subwindow to a new location. This function takes effect only when the waveform display is in the composite mode with only one Y axis displayed.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *l_origin* | List of two numbers in waveform coordinates that specify the new location for the axis origin. If *l_origin* is `nil`, the axis origin goes to the most logical position according to the data. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the axis origin is set. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSetOrigin( window(2) list( 4.5 5.5 ) ?subwindow 3 )
```

For subwindow 3, draws the X axis at Y = 5.5 and the Y axis at X = 4.5.

## awvSetPlotStyle

```
awvSetPlotStyle(
     w_windowId
     S_style
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Sets the plotting style for all the waveforms in a subwindow. If the plotting style is *bar* and the display mode is *smith*, then the plotting style is ignored until the display mode is set to *strip* or *composite.*

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *S_style* | Plotting style for the subwindow. Valid Values: `auto`, `scatterPlot`, `bar`, or `joined` |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the plotting style is set. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSetPlotStyle( window(2) "joined" ?subwindow 1 )
```

In subwindow 1, joins the adjacent points of the waveforms plotted with straight lines.

### Related Function

To get the plotting style for the waveforms in a subwindow, see the awvGetPlotStyle function on page-87.

## awvSetSmithModeType

```
awvSetSmithModeType(
     w_windowId
     t_type
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Sets the Smith display mode type for a subwindow. The display mode type takes effect only when the subwindow is set to *fs*.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *t_type* | Type of Smith display. Valid Values: `impedance`, `admittance`, or `polar` |
| *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the Smith display is set. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSetSmithModeType( window(2) "polar" ?subwindow 3 )
=> t
```

Sets the Smith display of subwindow 3 to polar.

### Related Functions

To return the Smith display type of a subwindow, see the awvGetSmithModeType function on page-91.

To get the display mode of a subwindow, see the awvGetDisplayMode function on page-81.

To set the display of a subwindow to a different type, see the awvSetDisplayMode function on page-182.

## awvSetSmithXLimit

```
awvSetSmithXLimit(
     w_windowId
     l_minMax
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Set the X axis display limits for a subwindow with a Smith display mode. This command does not take effect if the display mode is set to *strip* or *composite*.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *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 *autoscale*. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the X axis display limits are set. |
| `nil` | Returns `nil` if the Waveform window or subwindow do not exist. |

### Example

```
awvSetSmithXLimit( window(3) list(0 20) ?subwindow 2)
=> t
```

For subwindow 2, sets the X axis to display from 0 to 20.

### Related Functions

To set the Y axis display limits for a subwindow with a Smith display mode, see the awvSetSmithYLimit function on page-194.

To set the X axis display limits for a subwindow, see the awvSetXLimit function on page-201.

To set the Y axis display limits for a subwindow, see the awvSetYLimit function on page-205.

## awvSetSmithYLimit

```
awvSetSmithYLimit(
     w_windowId
     l_minMax
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Set the Y axis display limits for a subwindow with a Smith display mode. This command does not take effect if the display mode is set to *strip* or *composite*.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *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 autoscale. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the Y axis display limits are set. |
| `nil` | Returns `nil` if the Waveform window or subwindow do not exist. |

### Example

```
awvSetSmithYLimit( window(7) list(0 15) ?subwindow 2)
=> t
```

For subwindow 2, sets the Y axis to display from 0 to 15.

### Related Functions

To set the X axis display limits for a subwindow with a Smith display mode, see the awvSetSmithXLimit function on page-192.

To set the X axis display limits for a subwindow, see the awvSetXLimit function on page-201.

To set the Y axis display limits for a subwindow, see the awvSetYLimit function on page-205.

## awvSmithAxisMenuCB

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

### Description

Displays the *Axes (Smith Plot)* Option Menu.

The function is defined in `dfII/etc/context/awv.cxt`.

### Arguments

None

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the command was successful. |
| `nil` | Otherwise, returns `nil`. |

### Example

```
awvSmithAxisMenuCB() => t
```

## awvSetUpdateStatus

```
awvSetUpdateStatus(
     w_windowId
     g_enable
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Turns the update status of a subwindow *on* or *off*. If the update status is *off*, the subwindow display does not change when you tell the Waveform window to update the results.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *g_enable* | Boolean flag that specifies whether the update status is *on* or *off*. If this argument is set to `nil` and the update status of the specified subwindow is *on*, the update status is turned *off*. If this argument is set to non-`nil` and the update status of the subwindow is *off*, the update status is turned *on*. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the update status is turned on or off. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSetUpdateStatus( window(2) t ?subwindow 1 )
=> t
```

Turns on the update status for subwindow 1.

## awvSetWaveformDisplayStatus

```
awvSetWaveformDisplayStatus(
     w_windowId
     x_waveIndex
     g_enable
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Turns the display of a curve *on* or *off* based on the value of the `g_enable` flag. You can use this function to hide a curve without deleting it.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *x_waveIndex* | Integer identifying the waveform curve. (This is the integer identifier that you supplied in the `l_waveIndexList` argument in a previous Waveform window SKILL function.) |
| *g_enable* | Boolean flag that specifies whether the curve display is turned *on* or *off*. If this argument is set to `nil` and the specified curve is displayed, the curve display is removed. If this argument is set to non-`nil` and the specified curve is not displayed, the curve display is turned *on*. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the curve display is turned on or off. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSetWaveformDisplayStatus( window(2) 2 nil ?subwindow 1 )
=> t
```

Turns off the display of curve 2 in subwindow 1.

## awvSetWaveNameList

```
awvSetWaveNameList(
     list( list(trace_id1 trace_id2 ...)
     list(name1 name2 ...)
     )
     => t / nil
```

### Description

Sets the names of the list of waveform curves returned by the awvGetWaveNameList function.

### Arguments

| | |
|---|---|
| *trace_id* | The ID of a waveform curve returned by the awvGetWaveNameList function. |
| *name* | The name of the waveform curve to be used for a `trace_id`. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if successful. |
| `nil` | Returns `nil` otherwise. |

### Example

```
awvSetWaveNameList( list( list(1 2) list("wave1" "wave2"))
```

Sets the name of the waveform curve with the IDs `1` and `2` to `wave1` and `wave2` respectively.

## awvSetXAxisLabel

```
awvSetXAxisLabel(
     w_windowId
     g_label
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Lets you specify an X axis label to replace the automatically computed label.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *g_label* | Text for the X axis label. You can set this argument to `nil` to display the automatically computed label. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if successful. |
| `nil` | Returns `nil` otherwise. |

### Example

```
awvSetXAxisLabel( window(4) "Time in Seconds" ?subwindow 3)
=> t
```

Sets the X axis label of subwindow 3 to *Time in Seconds*.

### Related Functions

To return the current X axis label, see the awvGetXAxisLabel function on page-99.

To specify a Y axis label to replace the automatically computed label, see the awvSetYAxisLabel function on page-203.

## awvSetXLimit

```
awvSetXLimit(
     w_windowId
     l_minMax
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Sets the X axis display limits for a subwindow. This command does not take effect if the display mode is set to Smith.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *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 *autoscale.* |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the X axis display limits are set. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSetXLimit( window(2) list( 1 100 ) ?subwindow 3 )
```

For subwindow 3, sets the X axis to display from 1 to 100.

### Related Function

To set the Y axis display limits for the waveforms associated with a particular Y axis a subwindow, see the awvSetYLimit function on page-205.

## awvSetXScale

```
awvSetXScale(
     w_windowId
     t_scale
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Sets the display mode of the X axis in a subwindow. This command does not take effect if the display mode is set to Smith.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *t_scale* | Specifies the scale for the X-axis. Valid Values: `auto`, `log`, or `linear` |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the scale for the X axis is set. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSetXScale( window( 2 ) "log" ?subwindow 3 )
```

For subwindow 3, sets the X axis to display logarithmically.

### Related Function

To set the Y axis of a subwindow to display logarithmically, see the awvLogYAxis function on page-113.

## awvSetYAxisLabel

```
awvSetYAxisLabel(
     w_windowId
     x_yNumber
     g_label
     [ ?subwindow   x_subwindow ]
     [ ?stripNumber x_stripNumber ]
     )
     => t / nil
```

### Description

Lets you specify a Y axis label to replace the automatically computed label.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *x_yNumber* | Specifies the Y axis whose label you want to specify. Valid Values: 1 through 4 |
| *g_label* | Text for the Y axis label. You can set this argument to `nil` to display the automatically computed label. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |
| `?stripNumber` *x_stripNumber* | Specifies the strip containing the Y axis whose label you want to specify. Valid Value: Any number greater than 0. Default Value: 1 |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if successful. |
| `nil` | Returns `nil` otherwise. |

### Example

```
awvSetYAxisLabel( window(8) 2 "Voltage" ?subwindow 3 )
=> t
```

Sets the label display for Y axis 2 in subwindow 3 to *Voltage*.

### Related Functions

To return the current Y axis label, see the awvGetYAxisLabel function on page-102.

To specify an X axis label to replace the automatically computed one, see the awvSetXAxisLabel function on page-200.

## awvSetYLimit

```
awvSetYLimit(
     w_windowId
     x_yNumber
     l_minMax
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Sets the Y axis display limits for the waveforms associated with a particular Y axis and strip in a subwindow. If you do not specify *x_stripNumber*, the limits are applied when the subwindow is in *composite* mode.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *x_yNumber* | Specifies the Y axis to have the limited display. Valid Values: 1 through 4 |
| *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 *autoscale*. |
| `?stripNumber` *x_stripNumber* | Specifies the strip in which the Y display is to be limited. Valid Values: Integer value greater than or equal to 1 |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the Y axis display limits are set. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvSetYLimit( window(2) 1 list( 4.5 7.5 ) )
```

Sets the Y axis (Y1) to display from 4.5 to 7.5. This takes effect only in *composite* mode.

```
awvSetYLimit( window(2) 1 list( 4.5 7.5 ) ?stripNumber 2 )
```

Sets the Y axis (Y1) in strip 2 to display from 4.5 to 7.5. This takes effect only in *strip* mode.

### Related Function

To set the X axis display limits for a subwindow, see the awvSetXLimit function on page-201.

## awvSetYRange

```
awvSetYRange(
     w_windowId
     x_yNumber
     n_range
     [ ?subwindow   x_subwindow ]
     [ ?stripNumber x_stripNumber ]
     )
     => t / nil
```

### Description

Sets the range for the Y axis.

The range is the difference between the maximum Y value and the minimum Y value, and the range must be positive. The maximum value for the Y axis is automatically computed according to the waveforms. The minimum is calculated with the range argument.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *x_yNumber* | Specifies the Y axis whose range you want to set. Valid Values: 1 through 4 |
| *n_range* | Amount to subtract from the maximum value to determine the minimum value (for the Y axis). |
| ?subwindow *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |
| ?stripNumber *x_stripNumber* | Specifies the strip containing the Y axis whose range you want to set. |

### Value Returned

| | |
|---|---|
| t | Returns t when the range is set. |
| nil | Returns nil otherwise. |

### Example

```
awvSetYRange( window(3) 2 10 ?subwindow 4 )
=> t
```

Specifies that the difference between the maximum Y axis value and the minimum Y axis for Y axis 2 is 10.

### Related Function

To set the Y axis display limits for the waveforms associated with a particular Y axis of a subwindow, see the awvSetYLimit function on page-205.

## awvSimplePlotExpression

```
awvSimplePlotExpression(
     w_windowId
     t_expr
     l_context
     g_replace
     [ ?expr        l_dispExprList ]
     [ ?subwindow   x_subwindow ]
     [ ?yNumber     l_yNumberList ]
     [ ?stripNumber l_stripNumberList ]
     [ ?showSymbols l_showList ]
     [ ?lineStyle   l_styleList ]
     [ ?color       l_colorlist ]
     [ ?lineThickness l_thicknessList ]
     )
     => t / nil
```

### Description

If *g_replace* is set to t, this function evaluates the *t_expr* expression and gives the resulting waveforms the lowest numbers already assigned to existing curves.

The old curves are overwritten. Otherwise, the resulting curves are given new numbers, which are the next lowest and in sequence, and the old curves are not overwritten.

You can provide an optional list of strings, *l_dispExprList*, with this function call. When you supply this list, the strings are displayed in the Waveform window instead of the expressions.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *t_expr* | String containing an expression that is evaluated once when the command is issued and reevaluated at the completion of each simulation in the auto-update mode. *t_expr* can be `nil`. |
| *l_context* | Data context for a particular simulation. If evaluating the expressions requires data generated during a previously saved simulation or requires simulation data that is being generated in real time, you must specify this argument. Otherwise, specify `nil`. |
| *g_replace* | Flag that specifies which numbers to give the resulting waveforms. Valid Values:<br>`non-nil`- Specifies that the waveforms are given the lowest numbers, even if the numbers are already assigned to existing curves. In this case, the existing curves are overwritten.<br>`nil` - Specifies that the resulting curves are given new numbers, which are the next lowest and in sequence. |
| `?expr` *l_dispExprList* | List of strings to display in the Waveform window instead of the expressions. |
| `?subwindow` *x_subwindow* | Number for the subwindow (found in the upper right corner). Default Value: Current subwindow |
| `?yNumber` *l_yNumberList* | List of numbers identifying the Y axes. |
| `?stripNumber` *l_stripNumberList* | List of numbers identifying the strips. |
| `?showSymbols` *l_showList* | Flag or list of flags that specifies if the symbols for the traces should be plotted or not. By default, the value is set as `nil` and none of the symbols is plotted. Set the value as `t` to show symbols for all the traces. To set the show status for each symbol, specify a list of flags, one flag for each symbol. The number of flags should match the number of symbols in `l_symbolList`. Each flag specifies if the corresponding symbol in the `l_symbolList` list will be shown or not. Default Value: `nil` |
| `?lineStyle` *l_styleList* | Specifies the line-style of the signal. Valid values: `Solid`, `Dotted`, `Dashed`, `Dotdashed` |

| | |
|---|---|
| `?color` *`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`" |
| `?linethickness` *`l_thicknessList`* | Specifies the thickness of the signal. Valid values: `Fine`, `Medium`, `Bold` |

### Value Returned

| | |
|---|---|
| t | Returns `t` when *`t_expr`* is evaluated the resulting waveform curves are plotted. |
| nil | Returns `nil` if there is an error. |

### Example

```
awvSimplePlotExpression( window( 2 )
    "expr( x sin( x ) linRg( 0 1.5 .5 ) )" nil t )
```

Displays *sin(x)* from *x* = 0.0 to *x* = 1.5. The expression is evaluated at *x* = 0.0, 0.5, 1.0 and 1.5. *`g_replace`* is set to `t`, so the resulting waveforms are given the lowest numbers already assigned to existing curves, and the existing curves are overwritten.

### Related Functions

To evaluate an expression and assign the numbers specified in a list to the resulting waveforms, see the awvPlotExpression function on page-133.

To plot the Y values in a list against the X values in a list and display the resulting waveforms in a subwindow, see the awvPlotList function on page-141.

To plot the waveforms in a list in a subwindow, see the awvPlotWaveform function on page-151.

To evaluate an expression and add the resulting waveforms to a subwindow, see the awvAppendExpression function on page-35.

To plot the Y values in a list against the X values in a list and add the resulting waveforms to a subwindow, see the awvAppendList function on page-38.

To add the waveforms in a list to a subwindow, see the awvAppendWaveform function on page-41.

## awvTableSignals

```
awvTableSignals(
     l_siglist
     [ ?plotStyle t_plotStyle ]
     [ ?graphModifier t_graphModifier ])
     => t / nil
```

### Description

Displays a signal in the table window.

### Arguments

| | |
|---|---|
| *l_siglist* | List of signals to be plotted specified in the following format: (*list* (*list resultsDir1* (*list* (*list result1* (*list signal1 signal2* ...) ...) ...))) Remember to put quotation marks before and after each signal. |
| ?plotStyle *t_plotStyle* | Plot destination. Valid values: `Table`, `Append`, `Replace`, `New Window`, `New Subwindow`. Remember to put quotation marks before and after the style. |
| ?graphModifier *t_graphModifier* | X-axis of graph for rectangular graphs. Valid values: `Magnitude`, `Phase`, `WPhase`, `Real`, `Imaginary`, `dB10`, `dB20` |

### Value Returned

| | |
|---|---|
| `t` | If the signal is displayed in a table. |
| `nil` | Otherwise. |

### Example

```
awvTableSignals('((("./ampsim.raw" (("ac-ac" ("net10"))))) ?plotStyle "Append")
```

## awvUpdateAllWindows

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

### Description

Updates the display of all the Waveform Windows.

### Arguments

None.

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the Waveform Windows are updated. |
| `nil` | Returns `nil` if there are no Waveform Windows open. |

### Related Function

To update the display of only those subwindows whose update statuses are turned *on*, see the awvUpdateWindow function on page-214.

## awvUpdateWindow

```
awvUpdateWindow(
     w_windowId
     )
     => t / nil
```

### Description

Updates the display of all subwindows whose update-statuses are turned *on*.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the subwindows are updated. |
| `nil` | Returns `nil` if the specified Waveform window does not exist. |

### Example

```
awvUpdateWindow( window(4) )
=> t
```

Updates all the subwindows for the Waveform window.

### Related Functions

To turn the update status of a subwindow *on* or *off*, see the awvSetUpdateStatus function on page-197.

To return the list of subwindows (in a particular Waveform window) whose display and update statuses are on, see the awvGetOnSubwindowList function on page-86.

To update the display of all Waveform Windows, see the awvUpdateAllWindows function on page-213.

## awvZoomFit

```
awvZoomFit(
     w_windowId
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Returns the traces to its original size to fit in the window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the zoom fit operation is successful. |
| `nil` | Returns `nil` if the zoom fit operation fails. |

### Example

```
awvZoomFit(awvGetCurrentWindow())
awvZoomFit(awvGetCurrentWindow() ?subwindow 2)
```

## awvZoomGraphX

```
awvZoomGraphX(
     w_windowId
     l_minMax
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Zooms in or out the graph according to the specified X-axis (independent axis) coordinates.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *l_minMax* | List of two numbers in waveform coordinates that describe the minimum and maximum X-axis values for zoom. The first number is the minimum and the second is the maximum. The following notations are supported:<br>list(2e-7 2.4e-7)<br>2e-7:2.4e-7<br>200n:240n<br>list(200n 240n)<br>list("200ns" "240ns") |
| `?subwindow` *x_subwindow* | Number for the subwindow. Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the zoom in or out operation is successful. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvZoomGraphX(awvGetCurrentWindow() 200n:240n ?subwindow 2)
```

```
awvZoomGraphX(awvGetCurrentWindow() list("0.5us" "0.6us") ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
```

## awvZoomGraphY

```
awvZoomGraphY(
     w_windowId
     l_minMax
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Zooms in or out the graph according to the specified Y-axis (dependent axis) coordinates.

### Arguments

*w_windowId*
: Waveform window ID.

*l_minMax*
: List of two numbers in waveform coordinates that describe the minimum and maximum Y-axis values for zoom. The first number is the minimum and the second is the maximum. The following notations are supported:

  list(2e-7 2.4e-7)

  2e-7:2.4e-7

  200n:240n

  list(200n 240n)

  list("200ns" "240ns")

?stripNumber *x_stripNumber*
: Strip number to identify the strip on which zoom is performed. Default value: `1`

?subwindow *x_subwindow*
: Number for the subwindow. Default Value: Current subwindow

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the zoom in or out operation is successful. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvZoomGraphY(awvGetCurrentWindow() 1:10 ?subwindow 2 ?stripNumber 3)
```

```
awvZoomGraphY(awvGetCurrentWindow() list("1mv" "4.5mv") ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
```

## awvZoomGraphXY

```
awvZoomGraphXY(
     w_windowId
     l_xminMax
     l_yminMax
     [ ?stripNumber x_stripNumber]
     [ ?subwindow   x_subwindow ]
     )
     => t / nil
```

### Description

Zooms in or out the graph according to the specified X- and Y-axis coordinates.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. |
| *l_xminMax* | List of two numbers representing waveform X-axis coordinates that describe the limits for the zoom. The first number is the minimum and the second is the maximum. |
| *l_yminMax* | List of two numbers representing waveform Y-axis coordinates that describe the limits for the zoom. The first number is the minimum and the second is the maximum. |
| `?stripNumber` *x_stripNumber* | Strip number to identify the strip on which zoom is performed. Default value: `1` |
| `?subwindow` *x_subwindow* | Number for the subwindow. Default Value: Current subwindow |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` when the zoom in or out operation is successful. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvZoomGraphXY(awvGetCurrentWindow() 0:3 nil ?subwindow 2)
```

```
awvZoomGraphXY(awvGetCurrentWindow() 200n:240n list("1mv" "4.5mv") ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()) ?stripNumber 2)
```

## awvGetSubwindowTitle

```
awvGetSubwindowTitle(
     w_windowId
     [ ?subwindow w_subwindow ]
     )
     => t_subwindow / nil
```

### Description

Returns the title of the subwindow.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| *t_subwindow* | Returns the title of the specified subwindow. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwindow = awvGetCurrentSubwindow(window)
awvDisplaySubwindowTitle(window "My_subWindow" ?subwindow subwindow)
awvGetSubwindowTitle(window ?subwindow subwindow)=> "My_subWindow"
```

This example returns `My_subWindow` as the title of the current subwindow.

## awvGetWindowTitle

```
awvGetWindowTitle(
     w_windowId
     )
     => t_window / nil
```

### Description

Returns the title of the waveform window.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |

### Value Returned

| | |
|---|---|
| *t_window* | Returns the title of the specified window. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
awvDisplayTitle(window "My_Window" )
awvGetWindowTitle(window)
```

This example returns `My_Window` as the title of current waveform window.

## awvGetXAxisMajorDivisions

```
awvGetXAxisMajorDivisions(
     w_windowId
     [ ?subwindow w_subwindow ]
     )
     => x_majdivs / nil
```

### Description

Returns the number of major divisions that are set on the X-axis of a given graph.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| *x_majdivs* | Returns the number of major divisions on X-axis for the specified graph. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwindow = awvGetCurrentSubwindow(window)
awvGetXAxisMajorDivisions(window ?subwindow subwin)
```

This example returns the number of major divisions for the current subwindow.

## awvGetXAxisMinorDivisions

```
awvGetXAxisMinorDivisions(
     w_windowId
     [ ?subwindow w_subwindow ]
     )
     => x_mindivs / nil
```

### Description

Returns the number of minor divisions on the X-axis of a given graph.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| *x_majdivs* | Returns the number of major divisions on X-axis for the specified graph. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwindow = awvGetCurrentSubwindow(window)
awvGetXAxisMinorDivisions(window ?subwindow subwin)
```

This example returns the number of minor divisions for the current subwindow.

## awvGetXAxisStepValue

```
awvGetXAxisStepValue(
     w_windowId
     [ ?subwindow w_subwindow ]
     )
     => x_stepSize / nil
```

### Description

Returns the step size value for the X-axis for a given graph.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| *x_stepSize* | Returns the step size on X-axis for the specified graph. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwindow = awvGetCurrentSubwindow(window)
awvGetXAxisStepValue(window ?subwindow subwin)
```

This example returns the X-axis step size value for the current subwindow.

## awvGetXAxisUseStepValue

```
awvGetXAxisUseStepValue(
     w_windowId
     [ ?subwindow w_subwindow ]
     )
     => t / nil
```

### Description

Determines whether the X-axis scale uses the step value.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the step value has been used. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
awvGetXAxisUseStepValue(window ?subwindow subwin)
```

## awvSetXAxisMajorDivisions

```
awvSetXAxisMajorDivisions
     w_windowId
     x_numMajDiv
     [ ?subwindow w_subwindow ]
     )
     => t / nil
```

### Description

Sets the number of major divisions on the X-axis for the specified graph.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| *x_numMajDiv* | Number of major divisions to be set. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwin = awvGetCurrentSubwindow(window)
numMajDiv = 8
awvSetXAxisMajorDivisions(window 8 ?subwindow subwin)
```

This function sets 8 number of major divisions for the current subwindow.

## awvSetXAxisMinorDivisions

```
awvSetXAxisMinorDivisions(
     w_windowId
     x_numMinDiv
     [ ?subwindow w_subwindow ]
     )
     => t / nil
```

### Description

Sets the number of major divisions on the X-axis for the specified graph.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| *x_numMinDiv* | Number of minor divisions to be set. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwindow = awvGetCurrentSubwindow(window)
numMajDiv = 8
awvSetXAxisMinorDivisions(window 5 ?subwindow subwin)
```

This function sets 5 number of minor divisions for the current subwindow.

## awvSetXAxisStepValue

```
awvSetXAxisStepValue(
     w_windowId
     x_stepValue
     [ ?subwindow w_subwindow ]
     )
     => t / nil
```

### Description

Sets the step value for a X-axis for the specified graph.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| *x_stepValue* | Step size to be set. It must be a valid number. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
xAxisStepSize = 25n
subwin = awvGetCurrentSubwindow(win)
awvSetXAxisStepValue(win xAxisStepSize ?subwindow subwin)
```

This example sets the step value as `25n` for the X-axis in the current subwindow.

## awvSetXAxisUseStepValue

```
awvSetXAxisUseStepValue(
     w_windowId
     g_value
     [ ?subwindow w_subwindow ]
     )
     => t / nil
```

### Description

Specifies whether the step size value is to be used on the X-axis scale.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| *g_value* | Specifies whether step size value is used or not. Valid Values: `t` or `nil`. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwin = awvGetCurrentSubwindow(window)
awvSetXAxisStepValue(window 25n ?subwindow subwin)
```

The above example sets the X-axis step size to `25n`.

```
awvSetXAxisUseStepValue(window t ?subwindow subwin)
```

The above example sets the specified step value for the X-axis pertaining to specified graph.

```
awvSetXAxisUseStepValue(window nil ?subwindow subwin
```

The above example sets the default step size computed for the X-axis pertaining to specified graph.

## awvGetYAxisMajorDivisions

```
awvGetYAxisMajorDivisions(
     w_windowId
     x_yNumber
     [ ?subwindow w_subwindow ]
     )
     => x_majdivs / nil
```

### Description

Returns the number of major divisions that are set on the Y-axis of a given graph.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| *x_yNumber* | Specifies the Y-axis whose minor divisions to be returned. Valid Values: 1 through 4 |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| *x_majdivs* | Returns the number of major divisions on Y-axis for the specified graph. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwin = awvGetCurrentSubwindow(window)
awvGetYAxisMajorDivisions(window 1 ?subwindow subwin ?stripNumber 1)
```

This example returns the number of major divisions for the first Y-axis in the current subwindow.

## awvGetYAxisMinorDivisions

```
awvGetYAxisMinorDivisions(
     w_windowId
     x_yNumber
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow w_subwindow ]
     )
     => x_mindivs / nil
```

### Description

Returns the number of minor divisions on the specified Y-axis of a given graph.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| *x_yNumber* | Specifies the Y-axis whose minor divisions to be returned. Valid Values: 1 through 4 |
| `?stripNumber` *x_stripNumber* | Specifies the strip number for which minor divisions are to be returned. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| *x_majdivs* | Returns the number of major divisions on Y-axis for the specified graph. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwin = awvGetCurrentSubwindow(window)
awvGetYAxisMinorDivisions(window 2 ?subwindow subwin ?stripNumber 1)
```

This example returns the number of minor divisions for the Y-axis number 2 in the current subwindow.

## awvGetYAxisStepValue

```
awvGetYAxisStepValue(
     w_windowId
     x_yNumber
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow w_subwindow ]
     )
     => x_stepSize / nil
```

### Description

Returns the step size value for the specified Y-axis for a given graph.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| *x_yNumber* | Specifies the Y-axis whose for which the step size is to be returned. Valid Values: 1 through 4 |
| `?stripNumber` *x_stripNumber* | Specifies the strip number for which step size is to be returned. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| *x_stepSize* | Returns the step size on Y-axis for the specified graph. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwin = awvGetCurrentSubwindow(window)
awvGetYAxisStepValue(window 2 ?subwindow subwin ?stripNumber 4))
```

This example returns the Y-axis step size value for the strip number 4 of the current subwindow.

## awvGetYAxisUseStepValue

```
awvGetYAxisUseStepValue(
     w_windowId
     x_yNumber
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow w_subwindow ]
     )
     => t / nil
```

### Description

Determines whether the Y-axis scale use the step value for the specified strip in a given graph.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| *x_yNumber* | Specifies the Y-axis for which step size is to be set. Valid Values: 1 through 4 |
| `?stripNumber` *x_stripNumber* | Specifies the strip number for which the step size is to be set. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the step value has been used. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwin = awvGetCurrentSubwindow(window)
awvGetYAxisUseStepValue(window 2 ?subwindow subwin ?stripNumber 3)
```

This example returns true if step size has been used for strip number 3 in the current subwindow.

## awvSetYAxisMajorDivisions

```
awvSetYAxisMajorDivisions(
     w_windowId
     x_yNumber
     x_numMajDiv
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow w_subwindow ]
     )
     => t / nil
```

### Description

Sets the number of major divisions on the X-axis for the specified graph.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| *x_yNumber* | Specifies the Y-axis for which major divisions are to be set. Valid Values: 1 through 4 |
| *x_numMajDiv* | Number of major divisions to be set. |
| `?stripNumber` *x_stripNumber* | Specifies the strip number for which major divisions are to be set. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwin = awvGetCurrentSubwindow(window)
awvSetYAxisMajorDivisions(window 1 6 ?subwindow subwin ?stripNumber 6)
```

This function sets 6 number of major divisions for the strip number 6 in the current subwindow.

## awvSetYAxisMinorDivisions

```
awvSetYAxisMinorDivisions(
     w_windowId
     x_yNumber
     x_numMajDiv
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow w_subwindow ]
     )
     => t / nil
```

### Description

Sets the number of major divisions on the specified Y-axis for the specified graph.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| *x_yNumber* | Specifies Y-axis for which minor divisions are to be set. Valid Values: 1 through 4 |
| *x_numMajDiv* | Number of minor divisions to be set. |
| `?stripNumber` *x_stripNumber* | Specifies the strip number for which minor divisions are to be set. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwin = awvGetCurrentSubwindow(window)
awvSetYAxisMinorDivisions(window 1 6 ?subwindow subwin ?stripNumber 1))
```

This function sets 6 number of minor divisions for the strip number 1 in the current subwindow.

## awvSetYAxisStepValue

```
awvSetYAxisStepValue(
     w_windowId
     x_yNumber
     x_stepValue
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow w_subwindow ]
     )
     => t / nil
```

### Description

Sets the step value for the specified Y-axis for the specified graph.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| *x_yNumber* | Specifies Y-axis for which the step size is to be set. Valid Values: 1 through 4 |
| *x_stepValue* | Step size to be set. It must be a valid number. |
| `?stripNumber` *x_stripNumber* | Specifies the strip number for which the step size is to be set. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwin = awvGetCurrentSubwindow(window)
awvSetYAxisStepValue(window 1 0.25)
```

This example sets the step value as `0.25n` for the Y-axis in the current subwindow.

## awvSetYAxisUseStepValue

```
awvSetYAxisUseStepValue(
     w_windowId
     x_yNumber
     g_value
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow w_subwindow ]
     )
     => t / nil
```

### Description

Specifies whether the step size value is to be used on the Y-axis scale.

### Arguments

| | |
|---|---|
| *w_windowId* | Waveform window ID. You can specify window ID as window(1) or specify the current window by using the `awvGetCurrentWindow` function. |
| *x_yNumber* | Specifies Y-axis for which the step size is to be used. Valid Values: 1 through 4 |
| *g_value* | Specifies whether step size to be used or not. Valid Values: `t` or `nil`. |
| `?stripNumber` *x_stripNumber* | Specifies the strip number for which the step size is to be set. |
| `?subwindow` *w_subwindow* | Subwindow ID. |

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
window = awvGetCurrentWindow()
subwin = awvGetCurrentSubwindow(window)
awvGetYAxisStepValue(window 1 ?subwindow subwin ?stripNumber 1)
```

# OT

```
OT (
     t_instName
     t_paramName
     )
     =>o_waveform / nil
```

## Description

Returns the specified device parameter of the given instance from the transient analysis data.

## Arguments

| | |
|---|---|
| *t_instName* | Name of the instance for which the device parameter is to be returned. |
| *t_paramName* | Name of the parameter to be returned. |

## Values Returned

| | |
|---|---|
| *o_waveform* | Returns the output waveform if the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

## Example

```
OT("/I8/M3" "gm")
```

This function returns the output waveform for the parameter `gm` of instance `/I8/M3`.

# OS

```
OS (
     t_instName
     t_paramName
     )
     =>o_waveform / nil
```

### Description

Returns the specified device parameter of the given instance from the DC analysis data.

### Arguments

| | |
|---|---|
| *t_instName* | Name of the instance for which the device parameter is to be returned. |
| *t_paramName* | Name of the parameter to be returned. |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns the output waveform if the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
OS("/I8/M3" "gm")
```

This function returns the output waveform for the parameter `gm` of instance `/I8/M3`.

## pvrfreq

```
pvrfreq(
     s_ana
     t_pos
     t_neg
     t_res
     [ freq x_freq ]
     )
     => o_waveform / nil
```

### Description

Returns the spectral power at a specified frequency or at all frequencies with the resistor and voltage on the given positive and negative nodes.

### Arguments

| | |
|---|---|
| *s_ana* | Analysis type or analysis name. The available analyses are `hb`, `pss`, `qpss`, `pac`, `hbac` and `qpac`.<br>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_res* | The resistance value.<br>Valid values: Any integer or floating point number |
| *freq x_freq* | Frequency for which you want to plot the results. It is an optional field.<br>Valid values: Any integer value<br>Default Value: `nil` |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns the waveform representing the spectral power on specified frequency or at all frequencies with resistor and voltage on the positive and negative nodes. |
| *nil* | Returns `nil` if there is an error. |

### Example

Consider the following example:

```
pvrfreq("hb" "/RFin" "/RFout" 2 20 )
```

In this example, the spectral power waveform is generated for the following values:

- Analysis type—`hb`
- Positive node—`/RFin`
- Negative node—`/RFout`
- Resistance—`2`
- Frequency—`20`

## pvifreq

```
pvifreq(
     s_ana
     t_pos
     t_neg
     t_branch1
     t_branch2
     [ freq x_freq ]
     )
     => o_waveform / nil
```

### Description

Returns the spectral power from voltage and current for a specified frequency list or at all frequencies.

### Arguments

| | |
|---|---|
| *s_ana* | Analysis type or analysis name. The available analyses are `hb`, `pss`, `qpss`, `pac`, `hbac` and `qpac`.<br>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. |
| *freq x_freq* | Frequency for which you want to plot the results. It is an optional field.<br>Valid values: Any integer value<br>Default Value: `nil` |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns a waveform representing the spectral power on spectral power from voltage and current for a specified frequency list or at all frequencies. |
| *nil* | Returns `nil` if there is an error. |

### Example

Consider the following example:

```
pvifreq("hb" "/RFin" "/RFout" "/V1/PLUS" "/V2/PLUS" 20)
```

In this example, the spectral power waveform is generated for the following values:

- Analysis type—`hb`
- Positive node—`/RFin`
- Negative node—`/RFout`
- Branch name 1—`/V1/PLUS`

- Branch name 2—`/V2/PLUS`
- Frequency—`20`

## firstVal

```
firstVal(
     o_waveform
     )
     => n_value / nil
```

### Description

Returns the first value from where the waveform starts on the X-axis.

### Arguments

| | |
|---|---|
| *o_waveform* | Input waveform for which the starting X-axis value is to be returned. |

### Values Returned

| | |
|---|---|
| *n_value* | Returns the first X-axis value if the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
waveform = drCreateWaveform( drCreateVec( 'double list( 1 2 4 7 9)) drCreateVec(
'double list( 15 21 28 45 89 )))
firstVal(waveform)
"1.0"
```

## lastVal

```
lastVal(
     o_waveform
     )
     => n_value / nil
```

### Description

Returns the last value at which the waveform ends on the X-axis.

### Arguments

| | |
|---|---|
| *o_waveform* | Input waveform for which the ending X-axis value is to be returned. |

### Values Returned

| | |
|---|---|
| *n_value* | Returns the last X-axis value if the function runs successfully. |
| `nil` | Returns `nil` if there is an error. |

### Example

```
waveform = drCreateWaveform( drCreateVec( 'double list( 1 2 4 7 9)) drCreateVec(
'double list( 15 21 28 45 89 )))
lastVal(waveform)
"9.0"
```

## valueAt

```
valueAt(
     o_waveform
     n_xValue
     [ ?extrapolate g_extrapolate ]
     )
     => n_yValue / nil
```

### Description

Returns the Y-axis value of a given waveform at the specified X-axis value.

### Arguments

| | |
|---|---|
| *o_waveform* | Input waveform for which the Y-axis value is to be returned. |
| *n_xValue* | The X-axis value at which the Y-axis value is to be returned. |
| ?extrapolate *g_extrapolate* | Boolean indicating whether you want to perform extrapolation. If When this argument is set to `t`, the functions works similar to the `value` function. Default value: `nil`. |

### Values Returned

| | |
|---|---|
| *n_yValue* | When the function runs successfully, returns the Y-axis value at the specified X-axis. |
| `nil` | Returns `nil` is there is an error. Also, returns `nil` when extrapolation is set to `nil` and the specified X-axis value is out of range. |

### Example

Consider the below example in which you run `valueAt` function on the following input waveform:

```
waveform = drCreateWaveform( drCreateVec( 'double list( 1 2 4 7 9)) drCreateVec(
'double list( 15 21 28 45 89 )))
```

Case 1: Run this function to return the Y-axis value when X-axis is `3`:

```
valueAt(waveform 3)
```

The function returns the Y-axis value, `24.5`.

Case 2: Run this function to return the Y-axis value when X-axis is `10`:

```
valueAt(waveform 10)
```

The function returns `nil` and a warning message is displayed because the specified X-axis is out of the range.

```
*Warning* : Index value out of range for waveform(nil)
```

Case 3:, When you set the `extrapolate` argument to `t` and rerun this function to return the Y-axis value at X-axis=`10`

```
valueAt(waveform 10 ?extrapolate t)
```

The function estimates and returns the probable Y-axis value: `89.0`

## eyeMask

```
eyeMask(
     o_waveform
     t_xUnit
     [ @rest l_vertices ]
     )
     => o_waveform / nil
```

### Description

Creates a custom eye mask on the given eye diagram at the specified units and vertices.

### Arguments

| | |
|---|---|
| *o_waveform* | Input eye diagram on which eye mask is to be created. |
| *t_xUnit* | Units in which the X-axis coordinates are specified.<br>Valid values: `UI` or `s` |
| `@rest l_vertices` | List specifying the coordinates of the eye mask. The coordinates must be ordered adjacently.<br><br>**Note:** You can also specify the coordinates of the eye mask using the `VAR` function. |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns a waveform showing the input eye diagram with the specified mask. |
| `nil` | Returns `nil` if there is an error. |

### Example 1

The example below plots the eye mask on the given eye diagram with the following units and vertices:

- waveform: `eyeDiagram(v("signal" ?result "tran")`
- xUnits: `s`
- vertices: `(1n, 0) (1n, 1) (1.5n, 1) (1.5n, 0)`

The resultant expression is:

```
eyeMask( eyeDiagram(v("signal" ?result "tran") 0.1n 200.0n 2.5n) "s" '(1n 0) '(1n
1) '(1.5n 1) '(1.5n 0))
```

### Example 2

The example below plots the eye mask on the given eye diagram with the following units. Note that the coordinates, (`a1`, `b1`), (`a2`, `b2`), (`a3`, `b3`), and (`a4`, `b4`), are expressed using the `VAR` function:

- waveform: `eyeDiagram(v("signal" ?result "tran") 0.0 1e-08 4.688e-10 ?autoCenter t)`
- xUnits: `UI`
- vertices: `(a1, b1) (a2, b2) (a3, b3) (a4, b4)`

The resultant expression is:

```
eyeMask(eyeDiagram(v("signal" ?result "tran") 0.0 1e-08 4.688e-10 ?autoCenter t)
"UI" VAR("a1") VAR("b1") VAR("a2") VAR("b2") VAR("a3") VAR("b3") VAR("a4")
VAR("b4"))
```

## eyeMaskViolationPeriodCount

```
eyeMaskViolationPeriodCount(
     o_eyeMaskWf
     )
     => n_periods / nil
```

### Description

Returns the number of periods that contains an eye mask violation.

### Arguments

| | |
|---|---|
| *o_eyeMaskWf* | Eye mask expression that shows an eye diagram with a mask overlaid on it. |

### Values Returned

| | |
|---|---|
| *n_periods* | Returns a scalar value showing number of periods that contain an eye mask violation. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider the example below in which you apply `eyeMaskViolationPeriodCount` function on the following eye mask:

```
eyeMask=eyeMask( eyeDiagram(v("signal" ?result "tran") 0.1n 200.0n 2.5n) "s" '(1n
0) '(1n 1) '(1.5n 1) '(1.5n 0))
```

where,

```
Eye diagram waveform=eyeDiagram(v("signal" ?result "tran") 0.1n 200.0n 2.5n)
XUnits= s
Eye mask coordinates=(1n 0) '(1n 1) '(1.5n 1) '(1.5n 0))
eyeMaskViolationPeriodCount( eyeMask( eyeDiagram(v("signal" ?result "tran") 0.1n
200.0n 2.5n) "s" '(1n 0) '(1n 1) '(1.5n 1) '(1.5n 0)))
```

When you run the `eyeMaskViolationPeriodCount` function, it returns the number of periods for which the eye diagram intersects with the given eye mask.

## eyeBERLeft

```
eyeBERLeft(
     o_waveform
     n_start
     n_end
     n_period
     n_threshold
     n_noOfBins
     )
     => o_waveform / nil
```

### Description

Calculates the left-side bit-error rate curve for the specified eye diagram.

### Arguments

| | |
|---|---|
| *o_waveform* | Eye diagram waveform |
| *n_start* | Start time of the signal used to create eye diagram |
| *n_end* | End time of the signal used to create eye diagram |
| *n_period* | Period of the eye diagram |
| *n_threshold* | Crossing threshold value |
| *n_noOfBins* | Number of bins used to create histograms of crossing data |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns the left-side bit error rate curve. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider the following example to plot the BER curves for an eye diagram generated from a signal from time 0 to 400u with period 80n.

```
eye = eyeDiagram(v("jitter" ?result "tran") 0.0 400.0u 80n ?autoCenter t)
eyeBERLeft(eye 0 400u 80n 0.5 100)
```

## eyeBERRight

```
eyeBERRight(
     o_waveform
     n_start
     n_end
     n_period
     n_threshold
     n_noOfBins
     )
     => o_waveform / nil
```

### Description

Calculates the right-side bit-error rate curve for the specified eye diagram.

### Arguments

| | |
|---|---|
| *o_waveform* | Eye diagram waveform |
| *n_start* | Start time of the signal used to create eye diagram |
| *n_end* | End time of the signal used to create eye diagram |
| *n_period* | Period of the eye diagram |
| *n_threshold* | Crossing threshold value |
| *n_noOfBins* | Number of bins used to create histograms of crossing data |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns the right-side bit error rate curve. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider the following example to plot the BER curves for an eye diagram generated from a signal from time 0 to 400u with period 80n.

```
eye = eyeDiagram(v("jitter" ?result "tran") 0.0 400.0u 80n ?autoCenter t)
eyeBERRight(eye 0 400u 80n 0.5 100)
```

## eyeBERLeftApprox

```
eyeBERLeftApprox(
     o_waveform
     n_start
     n_end
     n_period
     n_threshold
     n_noOfBins
     )
     => o_waveform / nil
```

### Description

Estimates the left-side bit-error rate curve for the input eye diagram beyond the output of `eyeBERLeft` by tail-fitting the left-side cross distribution.

### Arguments

| | |
|---|---|
| *o_waveform* | Eye diagram waveform |
| *n_start* | Start time of the signal used to create eye diagram |
| *n_end* | End time of the signal used to create eye diagram |
| *n_period* | Period of the eye diagram |
| *n_threshold* | Crossing threshold value |
| *n_noOfBins* | Number of bins used to create histograms of crossing data |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns the left-side bit error rate curve. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider the following example to plot the BER curves for an eye diagram generated from a signal from time 0 to 400u with period 80n.

```
eye = eyeDiagram(v("jitter" ?result "tran") 0.0 400.0u 80n ?autoCenter t)
eyeBERLeftApprox(eye 0 400u 80n 0.5 100)
```

## eyeBERRightApprox

```
eyeBERRightApprox(
     o_waveform
     n_start
     n_end
     n_period
     n_threshold
     n_noOfBins
     )
     => o_waveform / nil
```

### Description

Estimates the right-side bit-error rate curve for the input eye diagram beyond the output of `eyeBERRight` by tail-fitting the right-side cross distribution.

### Arguments

| | |
|---|---|
| *o_waveform* | Eye diagram waveform |
| *n_start* | Start time of the signal used to create eye diagram |
| *n_end* | End time of the signal used to create eye diagram |
| *n_period* | Period of the eye diagram |
| *n_threshold* | Crossing threshold value |
| *n_noOfBins* | Number of bins used to create histograms of crossing data |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns the right-side bit error rate curve. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider the following example to plot the BER curves for an eye diagram generated from a signal from time 0 to 400u with period 80n.

```
eye = eyeDiagram(v("jitter" ?result "tran") 0.0 400.0u 80n ?autoCenter t)
eyeBERRightApprox(eye 0 400u 80n 0.5 100)
```

# 3

# Results Browser Functions

This chapter describes the SKILL functions that apply to the Results Browser in the Virtuoso Visualization and Analysis XL tool.

## rdbLoadResults

```
rdbLoadResults(
     t_sessionName
     t_resultsDir
     ) => t / nil
```

### Description

Loads the simulation results located at *resultsDir* into the browser associated with the specified *sessionName*.

### Arguments

| | |
|---|---|
| *t_sessionName* | Name of the session in which the results are to be loaded in the browser. Remember to put quotation marks before and after the session name. |
| *t_resultsDir* | Path to the location of the simulation results. Remember to put quotation marks before and after the path name. |

### Value Returned

| | |
|---|---|
| `t` | If the simulation results are loaded. |
| `nil` | Otherwise. |

### Example

```
rdbLoadResults( "unbound" "./opamplib/ampTest/adexl/results/data/Interactive.267/
1/opamplib:ampTest:1/psf")
```

## rdbReloadResults

```
rdbReloadResults(
     t_sessionName
     t_resultsDir)
     => t / nil
```

### Description

Re-loads the simulation results located at *resultsDir* into the browser associated with the specified *sessionName*. It is desirable to reload results during the successive simulation runs.

### Arguments

| | |
|---|---|
| *t_sessionName* | ADE-L/XL `sessionName` or `unbound` if running the tool in standalone mode and the Results Browser window is open. |
| *t_resultsDir* | Path to the location of the simulation results. Remember to put quotation marks before and after the path name. |

### Value Returned

| | |
|---|---|
| `t` | If the simulation results are re-loaded. |
| `nil` | Otherwise. |

### Example

```
rdbReloadResults( "unbound" list("/usr1/export/amptest/simulation/ampTest/
spectre/config/psf"))
```

## rdbUnloadResults

```
rdbUnloadResults(
     t_sessionName
     t_resultsDir
     )
     => t / nil
```

### Description

Unloads the simulation results located at *resultsDir* from the browser associated with the specified *sessionName*. It is desirable to unload results during the successive simulation runs to reduce resource consumption and remove clutter from the browser.

### Arguments

| | |
|---|---|
| *t_sessionName* | ADE-L/XL `sessionName` or `unbound` if running the tool in standalone mode and the Results Browser window is open. |
| *t_resultsDir* | Path to the location of the simulation results. Remember to put quotation marks before and after the path name. |

### Value Returned

| | |
|---|---|
| `t` | If the simulation results are unloaded. |
| `nil` | Otherwise. |

### Example

```
rdbUnloadResults( "unbound" list("/usr1/export/amptest/simulation/ampTest/
spectre/config/psf"))
```

## rdbSetCurrentDirectory

```
rdbSetCurrentDirectory(
     t_sessionName
     t_path
     )
     => t / nil
```

### Description

The Results Browser associated with the specified *sessionName* navigates to the directory specified by *path*.

### Arguments

| | |
|---|---|
| *t_sessionName* | ADE-L/XL `sessionName` or `unbound` if running the tool in standalone mode and the Results Browser window is open. |
| *t_path* | Path to the simulation results hierarchy to be displayed. Remember to put quotation marks before and after the path name. |

### Value Returned

| | |
|---|---|
| `t` | If the simulation results are displayed. |
| `nil` | Otherwise. |

### Example

```
rdbSetCurrentDirectory ("unbound" "./simulation/opampTest/spectre/schematic/psf/
tran/V11")
```

## rdbWriteToFormat

```
rdbWriteToFormat(
     t_sessionName
     t_path
     t_format
     l_signals
     )
     => t / nil
```

### Description

Outputs the specified signals from the browser associated with the specified *sessionName*.

### Arguments

| | |
|---|---|
| *t_sessionName* | ADE-L/XL `sessionName` or `unbound` if running the tool in standalone mode and the Results Browser window is open. |
| *t_path* | Path to the signals that are to be output. Remember to put quotation marks before and after the path name. |
| *t_format* | Format in which the signals are to be output. Valid values: `CSV` (Comma-Separated text), `VCSV` (Virtuoso Comma-Separated text), `PSF`, `SST2`, `Matlab`, `Spectre` Remember to put quotation marks before and after the format. |
| *l_signals* | List of signals to be output specified in the following format: `(list (list resultsDir1 (list (list result1 (list signal1 signal2 ...) ...) ...)))`. If you do not specify a signal name in this argument, the command outputs all the signals that are selected in the Results Browser. |

### Value Returned

| | |
|---|---|
| `t` | If the signals are displayed. |
| `nil` | Otherwise. |

### Example

```
rdbWriteToFormat( "unbound" "/tmp/output.vcsv" "VCSV" '((("./simulation/opampTest/
spectre/schematic/psf" (("tran" ("V11.p"))))))
```

## rdbShowDialog

```
rdbShowDialog(
     t_sessionName
     t_path
     t_format
     signals
     )
     => t / nil
```

### Description

Display and hides dialog boxes associated with the browser.

### Arguments

| | |
|---|---|
| *t_sessionName* | ADE-L/XL `sessionName` or `unbound` if running the tool in standalone mode and the Results Browser window is open. |
| *t_path* | Path to the signals that are to be output. Remember to put quotation marks before and after the path name. |
| *t_format* | Format in which the signals are to be output. Valid values: `CSV` (Comma-Separated text), `VCSV` (Virtuoso Comma-Separated text), `PSF`, `SST2`, `Matlab`, `Spectre` Remember to put quotation marks before and after the format. |
| *signals* | List of signals to be output specified in the following format: `(list (list resultsDir1 (list (list result1 (list signal1 signal2 ...) ...) ...)))`. If you do not specify a signal name in this argument, the command outputs all the signals that are selected in the Results Browser. |

### Value Returned

| | |
|---|---|
| `t` | If the specified dialog boxes are displayed or hidden. |
| `nil` | Otherwise. |

### Example

```
rdbShowDialog(unbound browser findResults show ((pathList ./ampsim.raw)))
```

## vvDisplayBrowser

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

### Description

Invokes the Results Browser within a window.

### Value Returned

| | |
|---|---|
| `t` | If the browser is invoked. |
| `nil` | Otherwise. |

## vivaInitBindkeys

```
vivaInitBindkeys()
```

### Description

Initializes the Virtuoso Visualization and Analysis XL bindkeys called from `viva.ini` context initialization file.

### Value Returned

None.

# 4

# Callback Functions

This chapter describes the following callback functions in detail:

- awviEditMenuCB
- awviMakeActiveMenuCB
- awviPLoadMenuCB
- awviPSaveMenuCB
- awviPUpdateMenuCB
- awviShowOutputMenuCB

## awviEditMenuCB

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

### Description

Callback for the *Expressions->Edit* menu in the Results Display window. It brings up the Expressions Edit form.

### Arguments

None

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the command was successful. |
| `nil` | Otherwise, returns `nil`. |

### Example

```
awviEditMenuCB() => t
```

## awviMakeActiveMenuCB

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

### Description

Callback for the menu *Window->Make Active* in the Results Display window. It makes current window the active print window.

### Arguments

None

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the command was successful. |
| `nil` | Otherwise, returns `nil`. |

### Example

```
awviMakeActiveMenuCB() => t
```

## awviPLoadMenuCB

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

### Description

Callback for the menu *Window->Load State* in the Results Display window. It loads a saved state of print window.

### Arguments

None

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the command was successful. |
| `nil` | Otherwise, returns `nil`. |

### Example

```
awviPLoadMenuCB() => t
```

## awviPSaveMenuCB

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

### Description

Callback for the menu *Window->Save State* in the Results Display window. It saves the current state of print window.

### Arguments

None

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the command was successful. |
| `nil` | Otherwise, returns `nil`. |

### Example

```
awviPSaveMenuCB() =>t
```

## awviPUpdateMenuCB

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

### Description

Callback for the *Window->Update Results* menu in the Results Display window. It updates the data using the current window setup.

### Arguments

None

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the command was successful. |
| `nil` | Otherwise, returns `nil`. |

### Example

```
awviPUpdateMenuCB() => t
```

## awviShowOutputMenuCB

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

### Description

Callback for the menu *Info->Show Output* in the Results Display window. It displays the expressions printed in the print window.

### Arguments

None

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the command was successful. |
| `nil` | Otherwise, returns `nil`. |

### Example

```
awviShowOutputMenuCB() => t
```

## appendWaves

```
appendWaves(
     o_wave1
     o_wave2
     [ o_waveN ]
     )
     => o_waveform / nil
```

### Description

Appends a series of input waveforms in the X vector direction into a single output waveform.It is the users responsibility to insure that the input waveforms X vectors are in the proper sequence.

### Arguments

| | |
|---|---|
| *o_wave1* | The first input waveform. |
| *o_wave2* | The second input waveform. |
| *o_waveN* | The optional Nth input waveform. |

### Values Returned

| | |
|---|---|
| *o_waveform* | Creates an input waveform. |
| `nil` | Otherwise, returns `nil` |

### Example

```
appendWaves( VT("/out" "path_to_reference_directory")
VT("/out")) => o_waveform
```

## waveVsWave

```
waveVsWave(
     [ ?x o_wavex ]
     [ ?y o_wavey ]
     [ ?xName t_xName ]
     [ ?xUnits g_xUnits ]
     [ ?yName t_yName ]
     [ ?yUnits g_yUnits ]
     )
     => o_waveform / nil
```

### Description

Creates an output waveform where its Y vector values are the Y vector values of the ?y input waveform and its X vector values are the Yvector values of the ?x input waveform. When the specified input waveforms have different X-axes, this function performs the interpolation. You can also use this function to compare the Y-axis of a family of waveforms with the Y-axis of a single-leaf waveform.

### Arguments

| | |
|---|---|
| ?x *o_wavey* | The Y vector values of this input waveform will be used for the Y vector values of the output waveform. |
| ?y *o_wavex* | The Y vector values of this input waveform will be used for the X vector values of the output waveform. |
| ?xName *t_xName* | Name of the X-axis. |
| ?xUnits *g_xUnits* | Determine whether to show units on the X-axis. |
| ?yName *t_yName* | Name of the Y-axis. |
| ?yUnits *g_yUnits* | Determine whether to show units on the Y-axis. |

### Values Returned

| | |
|---|---|
| *o_waveform* | Creates an output waveform. |
| nil | Otherwise, returns nil. |

### Example

```
waveVsWave( VT("/out" "path_to_reference_directory")
VT("/out")) => o_waveform
```

# 5

# Calculator Functions

This chapter lists the SKILL functions for Waveform Calculator.

## armSetCalc

```
armSetCalc(
     s_name
     g_value )
     => g_value
```

### Description

Sets the calculator resource of the specified property to the specified value.

### Arguments

| | |
|---|---|
| *s_value* | Name of any valid expression. |
| *g_value* | Value of a valid browser resource. |

### Value Returned

| | |
|---|---|
| *g_value* | Value of the browser resource. |

### Example

```
armSetCalc( 'numStack 10 )
=> g_value
```

Sets the number of stacks to be displayed in the calculator to 10.

## calCalculatorFormCB

```
calCalculatorFormCB(
     [ ?bBoxSpec t_bBoxSpec ]
     [ ?iconPosition t_iconPosition ]
     )
     => t / nil
```

### Description

Opens a new calculator window, if not already open. If the window is open, it activates the window and brings it in the focus.

### Arguments

`?bBoxSpec`
*`t_bBoxSpec`*

`?iconPosition`
*`t_iconPosition`*

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the command was successful. |
| `nil` | Otherwise, returns nil |

### Example

```
calCalculatorFormCB()
```

## calCalcInput

```
calCalcInput(
     isl_keyword
     [ t_expression ]
     )
```

### Description

The calCalcInput function manipulates the buffer and stack contents and enters arbitrary expressions into the buffer. The syntax of this function and a brief description of the arguments and valid keywords is given below.

### Arguments

| | |
|---|---|
| *isl_keyword* | Integer, symbol, or list of symbols indicating the function keywords to be performed on the buffer or stack. |

### Table of Keywords

The valid keywords you can pass to the `calCalcInput` function are those functions seen on the keypad and are summarized as follows.

| Keyword | Function |
|---|---|
| 0-9 | Enter numerals in the buffer |
| eex | Puts the calculator in exponential mode and enters an *e* in the buffer |
| point | Enters a decimal point in the buffer |
| pi, twoPi, sqrt2, charge, degPerRad, boltzmann, epp0 | Enter the constant names in the buffer |
| clear | Clears the buffer |
| lastx | Recalls the contents of the lastx register to the buffer |
| clst | Clears the buffer and all the stack registers |
| up, dwn, xchxy (x<>y on the keyboard) | Perform the up, down, and x-exchange-y operations on the buffer and stack contents |
| enter | Places a copy of the current buffer expression in the stack |

| Keyword | Function |
|---|---|
| append (app on the keyboard) | Appends the first stack element to the contents of the buffer |
| multiply, divide, add, subtract | Perform the specified operation on the buffer and the first stack element |
| chs | Changes the sign of the buffer expression or exponent |
| power | Raises 10 to the power of the buffer expression |
| square | Squares the buffer expression |
| exp | Calculates *e* to the power of the buffer expression |
| abs | Calculates the absolute value of the buffer expression |
| int | Truncates the integer portion of the buffer expression |
| db10 | Calculates the dB magnitude of the buffer contents for a power expression |
| db20 | Calculates the dB magnitude of the buffer contents for a voltage or current expression |
| sqrt | Calculates the square root of the buffer expression |
| mag | Calculates the magnitude of an AC buffer expression |
| phase | Calculates the phase of an AC buffer expression |
| imag | Calculates the imaginary part of an AC buffer expression |
| real | Calculates the real part of an AC buffer expression |
| log10 | Calculates the base-10 logarithm of the buffer expression |
| ln | Calculates the base-*e* (natural) logarithm of the buffer expression |
| ytox | Raises the contents of the first stack element to the power of the contents of the buffer |
| expression | Specifies the expression string to place in the buffer (Provides an implicit enter for the current buffer expression.) |
| t_expression | Specifies the expression string to place in the buffer (Argument used only with the expression keyword) |

## calCreateSpecialFunction

```
calCreateSpecialFunction(
     [ ?formSym s_formSym ]
     [ ?formInitProc s_formInitProc ]
     [ ?formTitle t_formTitle ]
     [ ?formCallback t_formCallback ]
     [ ?envGetVal t_envGetVal ]
     )
     => t / nil
```

### Description

Encapsulates the initialization and display of forms required for a special function.

### Arguments

| | |
|---|---|
| `?formSym` *`s_formSym`* | Form symbol for the form. |
| `?formInitProc` *`s_formInitProc`* | Symbol describing the procedure that creates the form entries and form. |
| `?formTitle` *`t_formTitle`* | String describing the special function. This title is placed on the window border of the form. |
| `?formCallback` *`t_formCallback`* | Callback for the form that processes the form fields and enters the expression into the calculator buffer. |
| `?envGetVal` *`t_envGetVal`* | Gives the name of the partition for the "calculator" tool. The function is called using this variable to retrieve the value for all the form fields ( if set ) from the default environment. |

## calCreateSpecialFunctionsForm

```
calCreateSpecialFunctionsForm(
     s_formSym
     l_fieldList
     )
     => t
```

### Description

Registers the form and calls an hiCreateForm to create the form. The form title and callback are specified through the call to calCreateSpecialFunction.

The form is also created with g_unmapAfterCB set to *t* (see hiCreateForm).

### Arguments

| | |
|---|---|
| *s_formSym* | Form symbol for the form. |
| *l_fieldList* | List of fields in the form. |

## calGetBuffer

```
calGetBuffer()
     => t_buffer
```

### Description

Gets the expression constructed in the calculator buffer.

### Arguments

None.

### Value Returned

| | |
|---|---|
| *t_buffer* | Returns the expression stored in the calculator buffer if the command was successful. It throws an error if the calculator form has not been initialized. |

### Example

```
calGetBuffer()
```

## calSetBuffer

```
calSetBuffer(
     t_buffer
     )
     => nil
```

### Description

Sets the contents of the calculator buffer.

### Arguments

| | |
|---|---|
| *t_buffer* | Expression to be stored in the calculator buffer. |

### Value Returned

`nil`

### Example

```
calSetBuffer("VT('out')")
```

Sets the contents of the calculator buffer to `VT('out')`.

## calSetCurrentTest

```
calSetCurrentTest(
     t_testName
     )
     => t / nil
```

### Description

Informs the calculator of the current test. When an access function (such as vt) is used from the calculator, it will display the schematic associated with the current test.

The tests are displayed in a drop-down in the calculator, which allows you to change the current test.

### Arguments

| | |
|---|---|
| *t_testName* | Name of current test. |

### Value Returned

| | |
|---|---|
| `t` | If the current test is displayed |
| `nil` | Otherwise. |

### Example

```
calSetCurrentTest(opamplib:amptest:1)
```

## caliModeToggle

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

### Description

Toggles the algebraic or RPN specific buttons on the calculator form based on the current value of the ‘calculator mode’.

### Arguments

None.

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the command was successful. |
| `nil` | Otherwise, returns nil |

### Example

```
caliModeToggle()
```

## caliRestoreDefaultWindowSize

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

### Description

Restores the original size of the calculator form window while maintaining the same upper left coordinate of the current window position.

### Arguments

None.

### Value Returned

| | |
|---|---|
| `t` | Returns `t` if the command was successful. |
| `nil` | Otherwise, returns nil |

### Example

```
caliRestoreDefaultWindowSize()
```

## calRegisterSpecialFunction

```
calRegisterSpecialFunction(
     l_sfinfo
     t_sfname
     s_sfcallback
     )
     => l_sfinfo / nil
```

### Description

Registers the specified special function information if it is not already registered.

### Arguments

| | |
|---|---|
| *l_sfinfo* | Name of the form list ( t_sfname s_sfcallback ). |
| *t_sfname* | Name that appears in the Special Functions menu. |
| *s_sfcallback* | Symbol describing the name of the callback procedure for this special function. |

## calSpecialFunctionInput

```
calSpecialFunctionInput(
     t_sfname
     l_fields
     )
     => t_expression / nil
```

### Description

Checks the buffer and stack and processes the arguments defined under `l_fields` into the buffer expression.

### Arguments

| | |
|---|---|
| *t_sfname* | Name of the function. |
| *l_fields* | Ordered list of form field symbols that compose the special function argument list. If the special symbol 'STACK is encountered in the list of fields, the top stack expression is popped into the special function argument list. Currently, all argument fields specified must have associated values or an error message is generated. If the field type is cyclic or radio, double quotes are added around the field value in the special function argument list. |

### Value Returned

| | |
|---|---|
| *t_expression* | Returns the expression in the Buffer with the specified argument values. |
| `nil` | Returns nil if there is an error. |

### Example1

```
calSpecialFunctionInput('test '(from STACK to))
```

This example defines a new function `test` with the following arguments:

- *from*—Obtains the value that you have specified in the `from` argument.
- `STACK`—Pops out the expression from the top of the stack.
- *to*—Obtains the value that you have specified in the `to` argument.

Suppose *from*=`10`, *to*=`20`, Top of stack=`v("out" ?result "tran")`

And, Buffer contains the following value:

```
Buffer=v("in" ?result "tran")
```

This function outputs the following expression in the Buffer:

```
test(Buffer from, STACK to)
```

```
test(v("in" ?result "tran") 10 v("out" ?result "tran")20)
```

### Example2

```
calSpecialFunctionInput( 'test nil )
```

This example defines a new function `test` with no arguments. When you select this function, the function will be directly applied on the expression in the Buffer.

## expr

```
expr(
     var
     expr
     l_values
     )
     => o_waveform / nil
```

### Description

Evaluates the expression by setting each of the values in the ‘l_values’ list to the ‘var’ variable in the ‘expr’.

A waveform object is created with ‘l_values’ forming the x-vector and the evaluated ‘expr’ forming the y-vector.

### Arguments

| | |
|---|---|
| *var* | variable |
| *expr* | expression containing the variable |
| *l_values* | list of x-values vectors |

### Value Returned

| | |
|---|---|
| *o_waveform* | Returns a waveform object. |
| nil | Otherwise, returns nil |

### Example

```
expr( x  (x*100)  '(10 20 30))
```

This implies that a waveform object  with x-vector values as ( 10 20 30 ) and y-vector values as ( 1000 2000 3000 ).

## famEval

```
famEval(
     l_expression
     [ t_va1 ]
     [ s_value1 ]
     )
     => o_result
```

### Description

Evaluates the expression with the sweep variables set as specified. A waveform or a number is returned.

### Arguments

| | |
|---|---|
| *l_expression* | expression |
| *va1* | sweep variable name |
| *s_value1* | value of the sweep variable |

### Value Returned

| | |
|---|---|
| *o_result* | Returns a waveform object or a number depending upon the inputs. |

### Example

```
famEval( VT("out") ?len "100u" )
```

## vvDisplayCalculator

```
vvDisplayCalculator(
     [ t_expr ]
     )
     => t / nil
```

### Description

Invokes the calculator within a window. If an expression is specified, the expression is displayed in the buffer.

### Arguments

| | |
|---|---|
| *t_expr* | Expression to be displayed in the calculator buffer. This is an optional argument. Remember to put quotation marks before and after the expression. |

### Value Returned

| | |
|---|---|
| `t` | If the calculator is invoked. |
| `nil` | Otherwise. |

### Example

```
vvDisplayCalculator(vt(\/net10\))
```

## adtFFT

```
adtFFT(
     l_list
     )
     => l_result / nil
```

### Description

Calculates the fast Fourier transform (FFT) of the input list.

### Arguments

| | |
|---|---|
| *l_list* | Input list for which you want to generate the fast Fourier transform list. |

### Values Returned

| | |
|---|---|
| *l_result* | Returns the fast Fourier transform list of the given input values. |
| nil | Returns nil if there is an error. |

### Example

Consider the following example in which you run the adtFFT on the given list of numbers:

```
adtFFT(list(3 9 2 5 1))
```

## adtIFFT

```
adtIFFT(
     l_list
     )
     => l_result / nil
```

### Description

Calculates the inverse discrete Fourier transform of the input list.

### Arguments

| | |
|---|---|
| *l_list* | Input list for which you want to generate the inverse of the discrete Fourier transform list. |

### Values Returned

| | |
|---|---|
| *l_result* | Returns the inverse list of values of the discrete Fourier transform of the given list of signals. |
| nil | Returns nil if there is an error. |

### Example

Consider the following example in which you run the adtFFT on the given list of numbers and then run adtIFFT function on the FFT results.

```
adtIFFT( adtFFT( list( 3 9 2 5 1) ) )
```

## topLine

```
topLine(
     o_waveform
     )
     => n_value / nil
```

### Description

Returns the topline value of the specified transient waveform.

### Arguments

| | |
|---|---|
| *o_waveform* | Input waveform whose topline value is to be calculated. |

### Values Returned

| | |
|---|---|
| *n_value* | Returns the topline value of the specified waveform. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider a results database that includes the following signal plotted for a transient analysis:

```
v = v("2" ?result "tran")
```

Here, v is a transient waveform. When you run the `topLine` function on this waveform, the following output is displayed:

```
topLine(v)  => -0.02851707
```

## baseLine

```
baseLine(
     o_waveform
     )
     => n_value / nil
```

### Description

Returns the baseline value of the specified transient waveform.

### Arguments

| | |
|---|---|
| *o_waveform* | Input waveform whose baseline value is to be calculated. |

### Values Returned

| | |
|---|---|
| *n_value* | Returns the baseline value of the specified waveform. |
| nil | Returns nil if there is an error. |

### Example

Consider a results database that includes the following signal plotted for a transient analysis:

```
v = v("2" ?result "tran")
```

Here, v is a transient waveform. When you run the baseLine function on this waveform, the following output is displayed:

```
baseLine(v)  => 3.26806
```

## topBaseLine

```
topBaseLine(
     o_waveform
     )
     => l_list / nil
```

### Description

Returns the topline and baseline values of a given transient waveform.

### Arguments

| | |
|---|---|
| *o_waveform* | Input waveform whose topline and baseline values are to be calculated. |

### Values Returned

| | |
|---|---|
| *l_list* | Returns the list containing topline and baseline values. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider the below example in which you open a results from the transient analysis:

```
openResults("./nand2_ring.raw")
v = v("2" ?result "tran")
```

where, `v` is a transient waveform.

When you run this function, the topline and baseline values are returned.

```
topBaseLine(v) => (3.26806 -0.02851707)
```

## leafValue

```
leafValue(
     o_waveform
     )
     => n_value / nil
```

### Description

This function is used as a wrapper function around the value function, which simplifies the value function syntax and also removes errors that may occur when you send an expression from Calculator to ADE outputs.

### Arguments

| | |
|---|---|
| *o_waveform* | Input waveform whose topline and baseline values are to be calculated. |

### Values Returned

| | |
|---|---|
| *n_value* | Returns the list containing topline and baseline values. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider the following Calculator expression(s) and their corresponding expression(s) that are created in the Outputs Setup pane when you send them to ADE:

- Expression in Calculator:

  ```
  leafValue(VT("/OUT") "modelFiles" "nom" "temperature" 27 "VDD" 1.8 )
  ```

  Expression in ADE:

  ```
  VT("/OUT")
  ```

- Expression in Calculator:

  ```
  clip(leafValue(v("/net07" ?result "tran") "Design_Point" 1.0) 1e-07 3e-07)
  ```

  Expression in ADE:

  ```
  clip(v("/net07" ?result "tran") 1e-07 3e-07)
  ```

- Expression in Calculator:

```
waveVsWave(?x leafValue(v("/net07" ?result "tran") "Design_Point" 1) ?y
leafValue( v("/net06" ?result "tran") "Design_Point" 1 ))
```

Expression in ADE:

```
waveVsWave(?x v("/net07" ?result "tran") ?y v("/net06" ?result "tran") )
```

- Expression in Calculator:

```
awvCreateBus("xyz" list(leafValue(getData("/net4" ?result "tran")
"Design_Point" 1) leafValue(getData("/net3" ?result "tran") "Design_Point" 1)
leafValue(getData("/net2" ?result "tran") "Design_Point" 1)
leafValue(getData("/net1" ?result "tran") "Design_Point" 1)) "Binary")
```

Expression in ADE:

```
awvCreateBus("xyz" list(getData("/net4" ?result "tran") getData("/net3"
?result "tran") getData("/net2" ?result "tran") getData("/net1" ?result
"tran") ) "Binary")
```

## swapSweep

```
swapSweep(
     o_waveform
     t_sweepVar
     [ ?xValue n_xValue ]
     )
     => o_waveform / nil
```

### Description

Swaps the X-axis value with the specified sweep variable.

### Arguments

| | |
|---|---|
| *o_waveform* | Represents the output waveform whose X-axis values you want to swap. |
| *t_sweepVar* | Sweep variable with which the X-axis of the given waveform is to be swapped. For example, `CAP`, `temp`. |
| `?xValue` *n_xValue* | X-axis value at which the waveform is swapped with the specified sweep variable. |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns the output waveform that has X-axis value swapped with the specified sweep variable. |
| `nil` | Returns `nil` if there is an error. |

### Example

In the below example, consider a parametric data simulated using the sweep variables, `CAP` and `temp`. When you run the following commands:

```
openResults("./psf")
selectResult('tran)
outWave = swapSweep( getData("/net9") "CAP" 1.7e+07)
```

The output waveform, `outWave`, will have the X-axis values swapped with specified `CAP` variable.

## rfEdgePhaseNoise

```
rfEdgePhaseNoise(
     [ ?result t_result ]
     [ ?eventList l_eventList ]
     [ ?resultsDir t_resultsDir ]
     )
     => o_waveform | nil
```

### Description

Plots the instantaneous phase noise, conversion of jitter to phase noise, spectrum plots related to jitter. It is a direct plot function.

### Arguments

| | |
|---|---|
| `?result` *`t_result`* | `pnoise_pmjitter` or `hbnoise_pmjitter`. |
| `?eventList` *`l_eventList`* | A list of two values, where the first value is sweep parameter value and the second value is event time index. For example, `?eventList '(val1 val2)`<br><br>If a sweep does not exist, the value for this argument can be specified as `'(val)`, where val is a single value for the event time index<br><br>With sweeps, the values in this argument can be specified as:<br><br>`list(list())`<br><br>`'(list(val1 val2))` or,<br><br>`'(list(val1 val2) list(val3 val4) …)` |
| `?resultsDir` *`t_resultsDir`* | Results directory path. |

### Values Returned

| | |
|---|---|
| *`o_waveform`* | Returns the output waveform representing instantaneous phase noise, conversion of jitter to phase noise, spectrum plots related to jitter. |
| `nil` | Returns `nil` if there is an error. |

### Example

- Consider pnoise jitter or hbnoise jitter without sweep:

  ```
  rfEdgePhaseNoise(?result "pnoise_sample_pm0" ?eventList '0)
  ```

  The example above returns the edge phase noise at event time index=0.

- Consider pnoise jitter or hbnoise jitter with sweep:

  ```
  rfEdgePhaseNoise(?result "pnoise_sample_pm0" ?eventList '((5.0 1) (6.0 2))
  ```

  The example above includes two lists with sweep parameter = 5 and 6, and event time index = 1 and 2

  ```
  rfEdgePhaseNoise(?result "pnoise_sample_pm0" ?eventList '((5.0 1)))
  ```

  The example above includes one list with sweep parameter = 5 and event time index = 1.

## rfInputNoise

```
rfInputNoise(
     t_unit
     [ ?result t_noiseResultName ]
     )
     => o_waveform / nil
```

### Description

Returns the input noise waveform. This command is run on the results of the Spectre pss-pnoise and hb-hbnoise analyses.

### Arguments

| | |
|---|---|
| *t_unit* | Specifies the Y-axis unit. The available values are `V/sqrt(Hz)`, `V**2/Hz`, `dBc/Hz`, and `dBV/Hz`. |
| `?result` *t_noiseResultName* | Name of the results file alias in which the input noise waveform is saved. |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns output waveform representing the input noise. |
| `nil` | Returns `nil` if there is an error. |

### Example

The following example plots the input noise waveform saved in the results file `pnoise` with Y-axis units as `V**2/Hz`.

```
plot(rfInputNoise("V**2/Hz" ?result "pnoise"))
```

## rfOutputNoise

```
rfOutputNoise(
     t_unit
     t_noiseResultName
     [ ?noiseConvention t_noiseConventionType ]
     )
     => o_waveform / nil
```

### Description

Returns the input noise waveform. This command is run on the results of the Spectre pss-pnoise and hb-hbnoise analyses.

### Arguments

| | |
|---|---|
| *t_unit* | Specifies the Y-axis units. The available values are `V/sqrt(Hz)`, `V**2/Hz`, `dBc/Hz`, and `dBV/Hz`. |
| *t_noiseResultName* | Name of the results file alias in which the output noise waveform is saved. |
| `?noiseConvention` *t_noiseConventionType* | Specifies the noise convention type. This argument is used on the AM or PM noise results. The available values are `DSB` and `SSB`. |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns output waveform representing the output noise. |
| `nil` | Returns `nil` if there is an error. |

### Example

The following example plots the output noise waveform saved in the results file `pnoise` with Y-axis unit as `V**2/Hz`.

```
plot(rfOutputNoise("V**2/Hz" ?result "pnoise"))
```

The following example plots the output noise waveform saved in the results file `pnoise_pm` with Y-axis unit as `V**2/Hz` and noise convention type as `DSB`.

```
rfOutputNoise("V**2/Hz" ?result "pnoise_pm" ?noiseConvention "DSB")
```

## rfTransferFunction

```
rfTransferFunction(
     t_unit
     [ ?result t_noiseResultName ]
     )
     => o_waveform / nil
```

### Description

Returns the transfer function waveform. This function is run on the results of the Spectre pss-pnoise and hb-hbnoise analysis.

### Arguments

| | |
|---|---|
| *t_unit* | Specifies the Y-axis units. The available values are `V/V` and `dB`. |
| `?result` *t_noiseResultName* | Name of the results file alias in which the gain waveform is saved. |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns the output waveform representing the transfer function. |
| `nil` | Returns `nil` if there is an error. |

### Example

The following example plots the transfer function waveform saved in the results file `pnoise` withY-axis unit as `V/V`.

```
plot(rfTransferFunction("V/V" ?result "pnoise"))
```

## numConv

```
numConv(
     t_inNum
     t_format
     g_needPrefix
     )
     => t_outNum / nil
```

### Description

Converts an input number into a given format and returns the output.

### Arguments

| | |
|---|---|
| *t_inNum* | The input number in string format that you want to convert. |
| *t_format* | Type of format. Valid values: `bin`, `dec`, `hex`, and `oct`. |
| *g_needPrefix* | Determines whether to add a prefix in output to indicate its format. If you set this argument to `t`, a prefix is added to the output, such as `0b`, `0x`, or `0`. If set to `nil`, the output does not include a prefix. |

### Values Returned

| | |
|---|---|
| *t_outNum* | Returns the converted number in string format. |
| `nil` | Returns `nil` if there is an error. |

### Example

Converting a number into binary format, with *needPrefix* set to `t`:

```
numConv("0x10", "bin") returns "0b10000"
```

Converting a number into octal format, with *needPrefix* set to `nil`:

```
numConv("8", "oct") returns "010"
```

Converting a number into hexadecimal format, with *needPrefix* set to `t`:

```
numConv("16", "hex") returns "0x10"
```

Converting a number into decimal format, with *needPrefix* set to `nil`:

```
numConv("011", "dec") returns "9"
```

## busTransition

```
busTransition(
     o_waveform
     t_yFrom
     t_yTo
     [ n_nth ]
     [ t_xName ]
     )
     => t_nth / t_lastNth / t_allXvalue / nil
```

### Description

Return the time when a bus value is changed from one specified value to another specified value.

### Arguments

| | |
|---|---|
| *o_waveform* | Represents a bus waveform. |
| *t_yFrom* | The starting value from which the bus change is to be calculated. The prefixes are used to define the formats, such as 0b for binary, 0 for octal, 0x for hexadecimal. No prefix indicates decimal format. |
| *t_yTo* | The target value to which the bus is changed. The prefixes are used to define the formats, such as 0b for binary, 0 for octal, 0x for hexadecimal. No prefix indicates decimal format. |
| *n_nth* | (Optional) When n > 0, this function returns the nth X value when the bus is changed, when n<0, returns the last nth value. When n=0 (default value), returns the change of bus for all X values. |
| *t_xName* | (Optional) When nth=0, this argument determines the X units of the generated waveform. Valid values: `time` and `cycle`. Default value: `time` |

### Values Returned

| | |
|---|---|
| *t_nth* | Returns the nth value at which Y value changes when n>0. |
| *t_lastNth* | Returns the last nth X value at which Y value changes when n<0. |
| *t_allXValue* | Returns all X values at which Y value changes when n=0. |
| `nil` | Returns `nil` if the specified bus transition does not exist or if there is any error. |

### Examples

Consider the following examples:

- Finds the third time (nth=3) when a bus value changed from `0b11` to `0b00`.

```
xvec = drCreateVec('double '(1.1 2.2 3.3 4.4 5.5 6.6))
yvec = drCreateVec('busVec '("11" "00" "11" "00" "11" "00"))
wave = drCreateWaveform(dvec dvec)
busTransition(wave "0b11" "0b00" 3)
```

- Finds the last third time (nth=-3) when a bus value changed from `0b11` to `0b00`.

```
xvec = drCreateVec('double '(1.1 2.2 3.3 4.4 5.5 6.6))
yvec = drCreateVec('busVec '("11" "00" "11" "00" "11" "00"))
wave = drCreateWaveform(dvec dvec)
busTransition(wave "0b11" "0b00" -3)
```

- Finds all the time (nth=0) when a bus value changed from `0b11` to `0b00`.

```
xvec = drCreateVec('double '(1.1 2.2 3.3 4.4 5.5 6.6))
yvec = drCreateVec('busVec '("11" "00" "11" "00" "11" "00"))
wave = drCreateWaveform(dvec dvec)
busTransition(wave "0b11" "0b00" 0 'time)
```

## aaSP

```
aaSP(
     n_portOrder1
     n_portOrder2
     [ dataDirectory t_dataDirectory ]
     )
     => o_waveform / nil
```

### Description

Returns the S-parameter waveform for a two-port network for the specified portorder values. The S-parameters describe the response of an N-port network to voltage signals at each port.

### Arguments

| | |
|---|---|
| *n_portOrder1* | The first port number that is the responding port. |
| *n_portOrder2* | The second port number that is the incident port. |
| *dataDirectory*<br>*t_dataDirectory* | (Optional) The results directory. |

### Values Returned

| | |
|---|---|
| *o_waveform* | Returns the S-parameter waveform if the function runs successfully. |
| *nil* | Returns nil if there is an error. |

### Example

The following example returns the S11 parameter waveform. S11 means the response at port1 due to a signal at port 1.

```
s11 = aaSP( 1 1 dataDir )
```

## rfJitter

```
rfJitter(
     [ ?result t_result ]
     [ ?resultsDir t_resultsDir ]
     [ ?unit t_unit ]
     [ ?ber g_ber ]
     [ ?from n_from ]
     [ ?to n_to ]
     [ ?signalLevel t_signalLevel ]
     )
     => value / o_waveform / nil
```

### Description

Calculates jitter from the result of Pnoise sample (jitter) analysis. It is used to calculate Jee, JDelay, and RMS Phase Noise.

### Arguments

| | |
|---|---|
| `?result` *`t_result`* | Name of the result of Pnoise sample analysis. |
| *`?resultsDir t_resultsDir`* | Path of the results directory. |
| `?unit` *`t_unit`* | Unit of jitter measurement. Valid values are `ppm`, `Second`, and `UI`. |
| `?ber` *`g_ber`* | Value of bit-error rate (BER) when the signal level is peak-to-peak. |
| `?from` *`n_from`* | The lower frequency limiter. |
| `?to` *`n_to`* | The upper frequency limiter. |
| `?signalLevel` *`t_signalLevel`* | Signal level. Valid values are `peak-to-peak` and `rms`. |

### Values Returned

| | |
|---|---|
| *value* | Returns the jitter value if simulation is run in single run mode. |
| *o_waveform* | Returns the waveform of jitter if simulation is run in sweep mode. |
| `nil` | Returns `nil` if the result name is not correct and jitter cannot be calculated. |

### Example

The following example returns the jitter value if simulation is run in single run mode. It returns a waveform that shows sweep variable plotted on the X-axis and jitter value plotted on the Y-axis if simulation is run in sweep mode.

```
rfJitter(?result "pnoise_sample_pm0" ?unit "Second" ?from 1000 ?to 1000000
?signalLevel "rms")
```

## rfJc

```
rfJc(
     [ ?result t_result ]
     [ ?resultsDir t_resultsDir ]
     [ ?unit t_unit ]
     [ ?ber g_ber ]
     [ ?from n_from ]
     [ ?to n_to ]
     [ ?k n_k ]
     [ ?multiplier n_multiplier ]
     )
     => value / o_waveform / nil
```

### Description

Calculates cycle jitter from the result of Pnoise sample (jitter) analysis.

### Arguments

| | |
|---|---|
| `?result` *`t_result`* | Name of the result of Pnoise sample analysis. |
| *`?resultsDir t_resultsDir`* | Path of the results directory. |
| `?unit` *`t_unit`* | Unit of jitter measurement. Valid values are `ppm`, `Second`, and `UI`. |
| `?ber` *`g_ber`* | Value of bit-error rate (BER) when the signal level is peak-to-peak. |
| `?from` *`n_from`* | Lower frequency limiter. |
| `?to` *`n_to`* | Upper frequency limiter. |
| `?k` *`n_k`* | Number of cycles. |
| `?multiplier` *`n_multiplier`* | Frequency multiplier. The default value is `1`. |

### Values Returned

| | |
|---|---|
| *value* | Returns the value of cycle jitter if simulation is run in single run mode. |
| *o_waveform* | Returns the waveform of cycle jitter if simulation is run in sweep mode. |
| `nil` | Returns `nil` if the result name is not correct and jitter cannot be calculated. |

### Example

The following example returns the value of cycle jitter if simulation is run in single run mode. It returns a waveform that shows sweep variable plotted on the X-axis and jitter value plotted on the Y-axis if simulation is run in sweep mode.

```
rfJc(?result "pnoise_sample_pm0" ?unit "Second" ?from 1000 ?to 10000 ?k 1
?multiplier 1)
```

## rfJcc

```
rfJcc(
     [ ?result t_result ]
     [ ?resultsDir t_resultsDir ]
     [ ?unit t_unit ]
     [ ?ber g_ber ]
     [ ?from n_from ]
     [ ?to n_to ]
     [ ?k n_k ]
     [ ?multiplier n_multiplier ]
     )
     => value / o_waveform / nil
```

### Description

Calculates cycle-to-cycle jitter from the result of Pnoise sample (jitter) analysis.

### Arguments

| | |
|---|---|
| `?result` *`t_result`* | Name of the result of Pnoise sample analysis. |
| *`?resultsDir t_resultsDir`* | Path of the results directory. |
| `?unit` *`t_unit`* | Unit of jitter measurement. Valid values are `ppm`, `Second`, and `UI`. |
| `?ber` *`g_ber`* | Value of bit-error rate (BER) when the signal level is peak-to-peak. |
| `?from` *`n_from`* | Lower frequency limiter. |
| `?to` *`n_to`* | Upper frequency limiter. |
| `?k` *`n_k`* | Number of cycles. |
| `?multiplier` *`n_multiplier`* | Frequency multiplier. The default value is `1`. |

### Values Returned

| | |
|---|---|
| *value* | Returns the value of cycle-to-cycle jitter if simulation is run in single run mode. |
| *o_waveform* | Returns the waveform of cycle-to-cycle jitter if simulation is run in sweep mode. |
| `nil` | Returns `nil` if the result name is not correct and jitter cannot be calculated. |

### Example

The following example returns the value of cycle-to-cycle jitter if simulation is run in single run mode. It returns a waveform that shows sweep variable plotted on the X-axis and jitter value plotted on the Y-axis if simulation is run in sweep mode.

```
rfJcc(?result "pnoise_sample_pm0" ?unit "Second" ?from 1000 ?to 10000 ?k 1
?multiplier 1)
```

## rfThresholdXing

```
rfThresholdXing(
     [ ?result t_result ]
     [ ?resultsDir t_resultsDir ]
     )
     => o_waveform / nil
```

### Description

Calculates the threshold crossing value according to the jitter event time from the result of Pnoise or Hbnoise sample (jitter) analysis.

### Arguments

| | |
|---|---|
| `?result` *`t_result`* | Name of the result of Pnoise or Hbnoise sample analysis. |
| `?resultsDir` *`t_resultsDir`* | Path of the results directory. |

### Values Returned

| | |
|---|---|
| *`o_waveform`* | Returns the waveform of threshold crossing with a time range, and a marker on the waveform with a threshold value. |
| `nil` | Returns `nil` if the result name is not correct and jitter cannot be calculated. |

### Example

The following example returns a waveform that shows time plotted on the X-axis and threshold value plotted on the Y-axis, and a marker on the waveform.

```
rfThresholdXing(?result "pnoise_sample_pm0")
```

## rfWrlsCim3Value

```
rfWrlsCim3Value(s_probe)
     => f_cim3 / nil
```

### Description

Returns the value of third-order counter-intermodulation (CIM3).

### Arguments

| | |
|---|---|
| *s_probe* | Name of the probe for which CIM3 needs to be calculated. |

### Values Returned

| | |
|---|---|
| *f_cim3* | Returns the value of CIM3. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider the following example:

```
rfWrlsCim3Value("WPRB2")
=> -142.1461
```

## rfWrlsCim5Value

```
rfWrlsCim5Value(s_probe)
     => f_cim5 / nil
```

### Description

Returns the value of fifth-order counter-intermodulation (CIM5).

### Arguments

| | |
|---|---|
| *s_probe* | Name of the probe for which CIM5 needs to be calculated. |

### Values Returned

| | |
|---|---|
| *f_cim5* | Returns the value of CIM5. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider the following example:

```
rfWrlsCim5Value("WPRB2")
=> -141.4673
```

## rfWrlsMeasContour

```
rfWrlsMeasContour(
     t_sig
     [ ?maxValue n_maxValue ]
     [ ?minValue n_minValue ]
     [ ?numCont n_numCont ]
     [ ?closeCont g_closeCont ]
     [ ?modifier t_modifier ]
     )
     => n_family / nil
```

### Description

Returns the contours for the measurements of simulation results.

### Arguments

*t_sig* The waveform you want to analyze.

?maxValue *n_maxValue*

The largest value of the contour to be drawn.

?minValue *n_minValue*

The smallest value of the contour to be drawn.

?numCont *n_numCont*

The number of contours.

?closeCont *g_closeCont*

The boolean flag that specifies whether a closed or open contour is to be drawn. The value, `t`, indicates that a closed contour is to be drawn.

*?modifier t_modifier*

The unit modifier.

### Values Returned

*n_family* Returns the contours of the measurement with the number = *numCont*.

`nil` Returns `nil` if there is an error.

### Example

The following example returns `9` contours for MCP with `maxMCP` = `6`, `minMCP` = `3`, `modifier` = `dB20`, and closes the contours.

```
plot(rfWrlsMeasContour("WPRB0.mcp" ?maxValue 6 ?minValue 3 ?numCont 9 ?closeCont t
?modifier "dB20"))
```

## rfWrlsCcdfValues

```
rfWrlsCcdfValues(
     t_sig
     )
     => o_waveform / f_avgPower / f_peakPower / nil
```

### Description

Plots the CCDF curve or returns the average or peak power from the results of an ENVLP wireless simulation.

### Arguments

| | |
|---|---|
| *t_sig* | The value you want to plot or calculate.<br>For example: `WPRB2.ccdf`, `WPRB2.avgPower`, or `WPRB2.peakPower` |

### Values Returned

| | |
|---|---|
| *o_waveform* | Plots the CCDF curve if the argument *t_sig* = `WPRBx.ccdf`. |
| *f_avgPower* | Returns the average power if the argument *t_sig* = `WPRBx.avgPower`. |
| *f_peakPower* | Returns the peak power if the argument *t_sig* = `WPRBx.peakPower`. |
| `nil` | Returns `nil` if there is an error. |

### Example

The following example plots the CCDF curve measured by `wprobe2`.

```
plot(rfWrlsCcdfValues("WPRB2.ccdf"))
```

The following example plots the average power measured by `wprobe2`.

```
plot(rfWrlsCcdfValues("WPRB2.avgPower"))
```

The following example plots the peak power measured by `wprobe2`.

```
plot(rfWrlsCcdfValues("WPRB2.peakPower"))
```

## rfCimMcpValue

```
rfCimMcpValue(s_probe)
    => f_power / nil
```

### Description

Returns the main channel power value when counter-intermodulation (CIM) is selected in LTE symbol.

### Arguments

| | |
|---|---|
| *s_probe* | Name of the probe for which the main channel power (MCP) needs to be calculated. |

### Values Returned

| | |
|---|---|
| *f_power* | Returns the value of MCP. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider the following example:

```
rfCimMcpValue("WPRB2")
=> 7.130826
```

## rfGetMinDampFactor

```
rfGetMinDampFactor()
     => n_result / nil
```

### Description

Returns the lowest damping ratio for the loops identified in the Loop Finder (LF) analysis.

### Arguments

None

### Values Returned

| | |
|---|---|
| *n_result* | Returns the lowest damping ratio when the function runs successfully. |
| *nil* | Returns `nil` if there is an error. |

### Example

Consider an example in which LF results are open.

```
rfGetMinDampFactor()
  => 1e-3
```

## rfGetEventtimeIndex

```
rfGetEventtimeIndex(
     t_signal
     t_resultName
     t_index
     )
     => x_number / nil
```

### Description

Returns the event time of a signal for the specified index value.

### Arguments

| | |
|---|---|
| *t_signal* | Name of the signal whose event time is to be calculated. |
| *t_resultName* | Name of the results directory. |
| *t_index* | Index value. |

### Values Returned

| | |
|---|---|
| *x_number* | Returns the event time of the signal for the specified index value. |
| `nil` | Returns `nil` if there is an error. |

### Example

The following example returns the event time result of the signal `RFout` for the index value `0` from the `pac_sampled` results directory

```
rfGetEventtimeIndex("/RFout" "pac_sampled" 0)
=> 1.16671e-10
```

## eyeHeightAtXY

```
eyeHeightAtXY(
     o_eyeDiagram
     f_x
     f_y
     )
     => f_eyeHeight / nil
```

### Description

Calculates the eye height at the specified point (*x*,*y*) inside the eye diagram. Eye height is the difference of two intercepts made with the innermost traces of the eye in the Y-axis direction.

**Note:** The specified coordinates (*x*,*y*) must lie within the open eye whose height you want to calculate.

For more information about how eye height is calculated, see the eyeHeightAtXY calculator function.

### Arguments

| | |
|---|---|
| *o_eyeDiagram* | The eye diagram waveform that is used to calculate the eye height. |
| *f_x* | The X-axis value that is used to calculate the eye height. |
| *f_y* | The Y-axis value that is used to calculate the eye height. |

### Values Returned

| | |
|---|---|
| *f_eyeHeight* | Returns the eye height at the specified point (*x*,*y*) inside the eye diagram. |
| `nil` | Returns `nil` if there is an error. |

### Example

The following example calculates the eye height at the point, *x* = `70p` and *y* = `2.2`.

```
eyeHeightAtXY(eyeDiagram(v("/example_1" ?result "tran") 560p 5.000n 140p
?triggerPeriod 7e-11 ?autoCenter t) 70p 2.2)
  => 1.494594
```

## eyeWidthAtXY

```
eyeWidthAtXY(
     o_eyeDiagram
     f_x
     f_y
     )
     => f_eyeWidth / nil
```

### Description

Calculates the eye width at the specified point ($x$,$y$) inside the eye diagram. Eye width is the difference of two intercepts made with the innermost traces of the eye in the X-axis direction.

**Note:** The specified coordinates ($x$,$y$) must lie within the open eye whose width you want to calculate.

For more information about how eye width is calculated, see the eyeWidthAtXY calculator function.

### Arguments

| | |
|---|---|
| *o_eyeDiagram* | The eye diagram waveform that is used to calculate the eye width. |
| *f_x* | The X-axis value that is used to calculate the eye width. |
| *f_y* | The Y-axis value that is used to calculate the eye width. |

### Values Returned

| | |
|---|---|
| *f_eyeWidth* | Returns the eye width at the specified point ($x$,$y$) inside the eye diagram. |
| `nil` | Returns `nil` if there is an error. |

### Example

The following example calculates the eye width at the point, $x$ = `70p` and $y$ = `2.2`.

```
eyeWidthAtXY(eyeDiagram(v("/example_1" ?result "tran") 560p 5.000n 140p
?triggerPeriod 7e-11 ?autoCenter t) 70p 2.2 )
  => 2.388933e-11
```

## triggeredDelay

```
triggeredDelay(
     o_signal1
     o_signal2
     n_thresh1
     s_edgeType1
     n_thresh2
     s_edgeType2
     [ ?multiple g_multiple]
     [ ?nth x_nth]
     [ ?periodicity x_periodicity]
     [ ?tol1 n_tol1]
     [ ?tol2 n_tol2]
     [ ?xName s_xName]
     )
     => o_waveform / n_value / nil
```

### Description

Calculates the delay from the trigger point on the edge of a triggering signal to the next edge of the target signal.

For more information about how the delay is calculated, see the triggeredDelay calculator function.

### Arguments

*o_signal1* Name of the triggering signal.

*o_signal2* Name of the target signal to measure delay.

*n_thresh1* Threshold value for signal 1.

*s_edgetype1* Edge type for signal 1.

*n_thresh2* Threshold value for signal 2.

*s_edgeType2* Edge type for signal 2.

?multiple *g_multiple*

Specifies whether you want to retrieve only one occurrence of a delay event for the given waveform (single), or all occurrences of an overshoot for the given waveform, which you can later plot or print.

Possible values:

- `t`: Returns the waveform of measured delay starting from the nth edge.
- `nil`: Returns a single delay at the nth edge.

Default value: `t`

?nth *x_nth* Edge number of the triggering signal from which the delay is to be calculated.

Default value: `1`

?periodicity *x_periodicity*

Periodic interval for the triggering signal.

Default value: `1`

?tol1 *n_tol1* Tolerance value to detect the threshold crossings for the triggering signal.

Default value: `0.0`

?tol2 *n_tol2* Tolerance value to detect the threshold crossings for the target signal.

Default value: `0.0`

| | |
|---|---|
| `?xName` *`s_xName`* | Specifies whether you want to retrieve the delay data against trigger time, target time (or another X-axis parameter for non-transient data), or cycle.<br>Default value: `trigger`<br>Possible values:<br>■ `trigger`<br>■ `target`<br>■ `cycle` |

### Values Returned

| | |
|---|---|
| *`o_waveform`* | Returns a waveform if the value of *`g_mutiple`* is set to `t`. |
| *`n_value`* | Returns a numeric value if the value of *`g_mutiple`* is set to `nil`. |
| `nil` | Returns `nil` if there is an error in calculation. |

### Example

The following example returns the value of delay measured from the trigger point on the first rising edge of the triggering signal, `samphold`, to the next rising edge of the target signal, `v2net`.

```
triggeredDelay(v("samphold" ?result "tran") v("v2net" ?result "tran") 1.7 "rising"
2.9 "rising" ?multiple nil ?xName "trigger" )
  => 5.40054e-08
```

## mu

```
mu(
     o_s11
     o_s12
     o_s21
     o_s22
     )
     => o_waveform / nil
```

### Description

Returns the alternative stability factor that indicates the minimum distance between the origin of the unit Smith chart and the load unstable region.

**Note:** If `mu > 1`, the two-port network is unconditionally stable.

### Arguments

| | |
|---|---|
| *o_s11* | Waveform object representing the S-parameter `s11`. |
| *o_s12* | Waveform object representing the S-parameter `s12`. |
| *o_s21* | Waveform object representing the S-parameter `s21`. |
| *o_s22* | Waveform object representing the S-parameter `s22`. |

### Value Returned

| | |
|---|---|
| *o_waveform* | Returns the waveform object indicating the minimum distance between the origin of the unit Smith chart and the load unstable region. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider the following S-parameters:

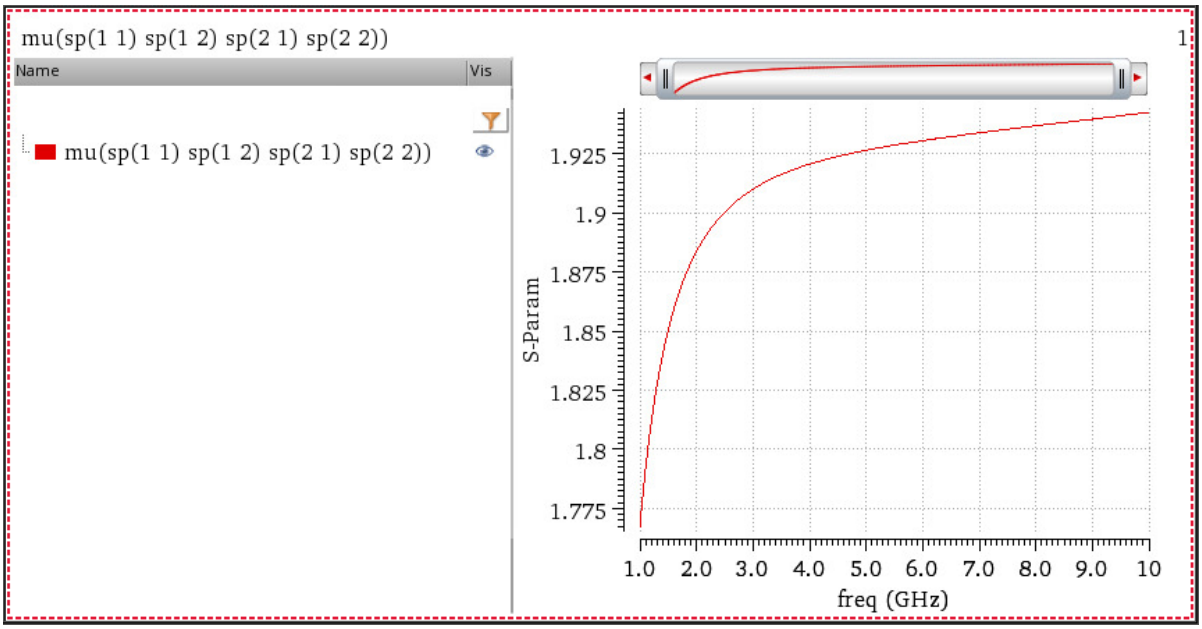
```
s11 = sp(1 1)
s12 = sp(1 2)
s21 = sp(2 1)
s22 = sp(2 2)
```

The `mu` function returns a waveform object:

```
mu(s11 s12 s21 s22)
=> srrWave:0x32e91540
```

To plot this waveform, run:

```
plot(mu(s11 s12 s21 s22))
```



# Mu

```
Mu(
     t_dataDir
     )
     => o_waveform / nil
```

## Description

Returns the alternative stability factor that indicates the minimum distance between the origin of the unit Smith chart and the load unstable region.

**Note:** If `mu > 1`, the two-port network is unconditionally stable.

## Argument

| | |
|---|---|
| *t_dataDir* | Path of the results directory that contains the results of S-parameter analysis. |

## Value Returned

| | |
|---|---|
| *o_waveform* | Returns the waveform object indicating the minimum distance between the origin of the unit Smith chart and the load unstable region. |
| `nil` | Returns `nil` if there is an error. |

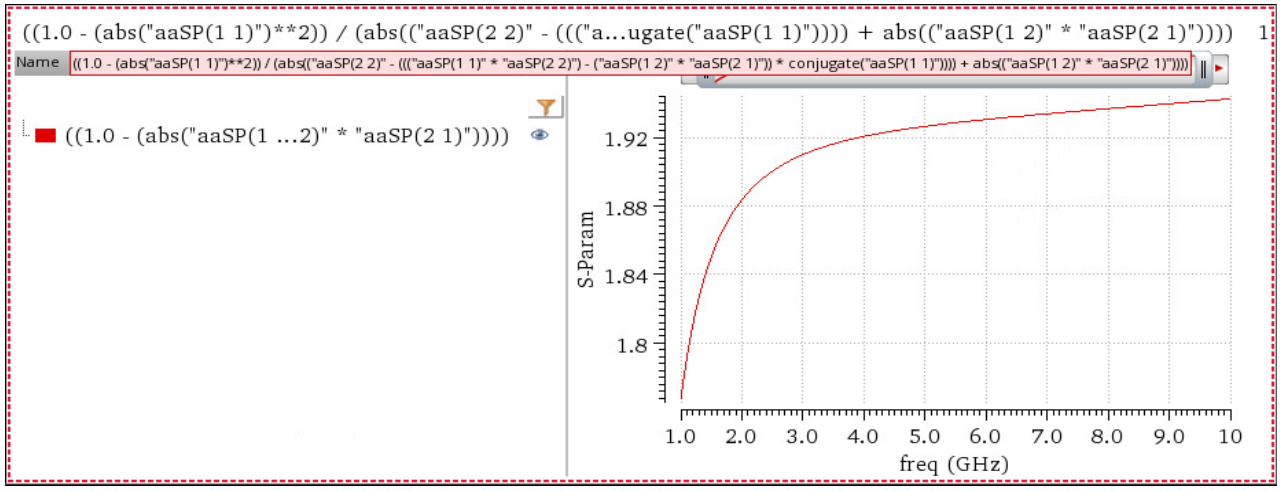
## Example

The following function returns a waveform object:

```
Mu("./simulation/ampTest/spectre/schematic/psf")
=> srrWave:0x32e91aa0
```

To plot this waveform, run:

```
plot(Mu("./simulation/ampTest/spectre/schematic/psf"))
```



## mu_prime

```
mu_prime(
     o_s11
     o_s12
     o_s21
     o_s22
     )
     => o_waveform / nil
```

### Description

Returns the alternative stability factor that indicates the minimum distance between the center of the unit Smith chart and the source unstable region.

### Arguments

| | |
|---|---|
| *o_s11* | Waveform object representing the S-parameter `s11`. |
| *o_s12* | Waveform object representing the S-parameter `s12`. |
| *o_s21* | Waveform object representing the S-parameter `s21`. |
| *o_s22* | Waveform object representing the S-parameter `s22`. |

### Value Returned

| | |
|---|---|
| *o_waveform* | Returns the waveform object indicating the minimum distance between the center of the unit Smith chart and the source unstable region. |
| `nil` | Returns `nil` if there is an error. |

### Example

Consider the following S-parameters:

```
s11 = sp(1 1)
s12 = sp(1 2)
s21 = sp(2 1)
s22 = sp(2 2)
```

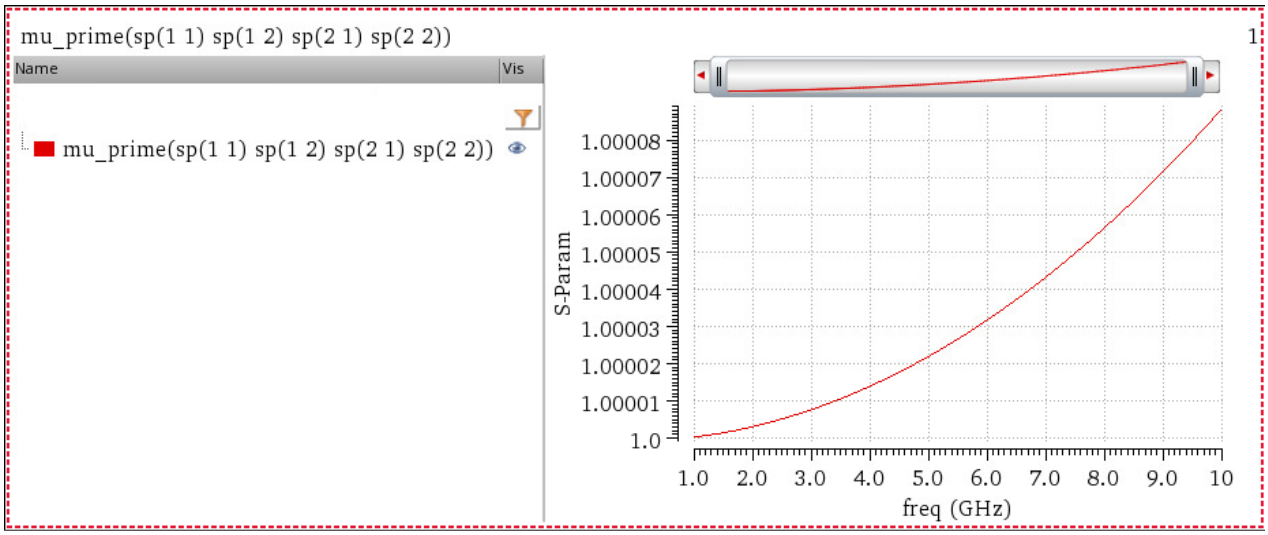
The `mu_prime` function returns a waveform object:

```
mu_prime(s11 s12 s21 s22)
```

```
=> srrWave:0x32e91780
```

To plot this waveform, run:

```
plot(mu_prime(s11 s12 s21 s22))
```



## Mu_prime

```
Mu_prime(
     t_dataDir
     )
     => o_waveform / nil
```

### Description

Returns the alternative stability factor that indicates the minimum distance between the center of the unit Smith chart and the source unstable region.

### Argument

| | |
|---|---|
| *t_dataDir* | Path of the results directory that contains the results of S-parameter analysis. |

### Value Returned

| | |
|---|---|
| *o_waveform* | Returns the waveform object indicating the minimum distance between the center of the unit Smith chart and the source unstable region. |
| `nil` | Returns `nil` if there is an error. |

### Example

The following function returns a waveform object:

```
Mu_prime("./simulation/ampTest/spectre/schematic/psf")
=> srrWave:0x328bc130
```

To plot this waveform, run:

```
plot(Mu_prime("./simulation/ampTest/spectre/schematic/psf"))
```