cadence®

Virtuoso Visualization and Analysis XL SKILL Reference

Product Version IC23.1
November 2023

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

Contents

4
Calculator Functions 383

1

Virtuoso Visualization and Analysis XL SKILL functions

Virtuoso Visualization and Analysis XL SKILL functions are grouped into following categories according to their purpose.

Category	Description
Waveform Window Functions	Lets you perform various operations on a subwindow of a Waveform window. These functions have the prefix `awv`.
Results Browser Functions	Lets you perform various operations on Results Browser. These functions have the prefix `rdb`.
Calculator Functions	Lets you perform various operations on Calculator. You can also perform various calculations on the results of transient and ac analysis and plot outputs using these functions.
RF Functions	Lets you perform various calculations on results of Spectre RF analyses and plot their outputs. Usually, these functions have the prefix `rf`.

Waveform Window Options

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 the `.cdsinit` file.

The following table describes various Waveform window options. Their default values are indicated in **`bold`** in the Valid Values column of the table.

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

Option	Description	Type	Valid Values
`displayAxes`	Displays axes.	Boolean	■ **`t`** ■ `nil`
`displayAxesBy125`	Displays the axis labels by increments of 1, 2, or 5.	Boolean	■ `t` ■ **`nil`**
`displayAxesLabel`	Displays axes labels.	Boolean	■ **`t`** ■ `nil`
`displayGrids`	Displays grid lines.	Boolean	■ `t` ■ **`nil`**
`displayMajorTicks`	Displays major tick marks.	Boolean	■ **`t`** ■ `nil`
`displayMinorTicks`	Displays minor tick marks.	Boolean	■ **`t`** ■ `nil`
`hcCopyNum`	Specifies the number of copies to plot.	integer	Any positive integer Default value: **`1`**
`hcDisplay`	Specifies the display name.	string	Defined in the technology file. Default value: "**`display`**"
`hcHeader`	Specifies header with plot.	Boolean	■ **`t`** ■ `nil`
`hcMailLogNames`	Emails plot submission output to user.	Boolean	■ **`t`** ■ `nil`
`hcOrientation`	Specifies the orientation of the plot.	string	■ "portrait" ■ "landscape" ■ "**`automatic`**"
`hcOutputFile`	Specifies to plot to the file only.	general (string or nil)	Name of the output file

Option	Description	Type	Valid Values
`hcPaperSize`	Specifies size of plot paper.	string	Specified in `.cdsplotinit`
`mode`	Controls mode type.	string	■ "strip" ■ "smith" ■ "**`composite`**"
`numIdentifier`	Controls the number of identifiers per waveform.	integer	Any positive integer or nil to show all the identifiers Default value: **`6`**
`style`	Controls how waveforms are plotted in the window.	string	■ "bar" ■ "scatterPlot" ■ "joined" ■ "**`auto`**"
`xLog`	Displays x axis logarithmically.	Boolean	■ **`t`** ■ `nil`

Related Topics

awvSetOptionDefault

awvSetOptionValue

Symbols for Data Points

You can choose to display symbols for data points on a waveform. These symbols are specified in a list using the argument `?dataSymbol`. The symbols for data points are displayed only when the argument `?showSymbols` is set to `t`.

The following table lists integers and characters that can be used to display their corresponding symbols.

Symbol	Integer	Character
+ (plus)	`0`, `10`, or `20`	"+"
. (dot)	`1`	"."
x (cross)	`2`	"`x`" or "`X`"
square	`5` or `15`	Not supported
circle	`4`, `6`, or `16`	"`o`" or "`O`"
box	`3`	Not supported

Related Topics

awvAppendExpression

awvAppendList

awvAppendWaveform

awvPlotExpression

awvPlotList

awvPlotWaveform

Calculator Keywords

The following table describes the keywords corresponding to the functions that can be performed on the buffer and stack contents.

Keyword	Description
`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`	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
`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

Keyword	Description
`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 $\log_{10}$ of the buffer expression
`ln`	Calculates the $\log_e$ 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.

2

Waveform Window Functions

The Waveform window SKILL functions let you perform various operations on a subwindow of a Waveform window. These functions have the prefix `awv`.

Each Waveform window can contain a number of subwindows. Each subwindow can be identified by an integer index. Note that even if a Waveform Window does not have any subwindows, the waveform display is still considered a subwindow. You can think of it as subwindow number one of one. A subwindow can contain a number of curves, which can be identified by an integer index.

When you work with the Waveform window using SKILL, you must use only the functions listed in this topic. 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.

This topic lists the Waveform window functions that are available in Virtuoso Visualization and Analysis XL.

awvGetXAxisStepValue

awvGetXAxisUseStepValue

awvGetXMarkerNames

awvGetYAxisLabel

awvGetYAxisMajorDivisions

awvGetYAxisMinorDivisions

awvGetYAxisStepValue

awvGetYAxisUseStepValue

awvGetYMarkerNames

awvInitWindowFunctionAdd

awvInitWindowFunctionDel

awvInitWindowFunctionGet

awvIsPlotWindow

awvLoadCustomCalcFunction

awvLoadEyeMask

awvLoadMenuCB

awvLoadSharedCustomFunctionsFile

awvLoadWindow

awvLogXAxis

awvLogYAxis

awvPlaceAMarker

awvPlaceBMarker

awvPlaceBookmark

awvPlaceCircleMarker

awvSetXAxisStepValue

awvSetXAxisUseStepValue

awvSetXLimit

awvSetXScale

awvSetYAxisLabel

awvSetYAxisMajorDivisions

awvSetYAxisMinorDivisions

awvSetYAxisStepValue

awvSetYAxisUseStepValue

awvSetYLimit

awvSetYRange

awvSetWaveNameList

awvSimplePlotExpression

awvSmithAxisMenuCB

awvTableSignals

awvUpdateAllWindows

awvUpdateWindow

awvZoomFit

awvZoomGraphX

awvZoomGraphXY

awvZoomGraphY

famGetExpr

famSetExpr

vvGetGraphBackground

vvSetGraphBackground

awvAddSubwindow

```
awvAddSubwindow(
     w_windowID
     )
     => x_subwindow / nil
```

Description

Adds a subwindow to the specified Waveform window.

Arguments

w_windowID	Waveform window ID.

Value Returned

x_subwindow	Identification number of the subwindow, which can be found in the top-right corner of the subwindow.
nil	The subwindow cannot be added because the specified Waveform window does not exist.

Examples

The following example creates a Waveform window and returns its window ID.

```
win=awvCreatePlotWindow()
=> window:3
```

The following example returns the identification number of the current subwindow in the specified Waveform window win.

```
awvGetCurrentSubwindow(win)
=> 1
```

The following examples adds a subwindow to the specified Waveform window and returns the identification number of the added subwindow. Note that the newly added subwindow is also selected as the current subwindow.

```
awvAddSubwindow(win)
=> 2
```

The following example returns the identification number of the current subwindow in the specified Waveform window win.

```
awvGetCurrentSubwindow(win)
=> 2
```

The following example returns the window ID of the current Waveform window.

```
window=awvGetCurrentWindow()
=> window:3
```

The following example adds a subwindow to the specified Waveform window.

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

Related Topics

awvCreatePlotWindow

awvGetCurrentSubwindow

awvGetCurrentWindow

awvAnalog2Digital

```
awvAnalog2Digital(
     o_waveform
     n_vhi
     n_vlo
     n_vc
     n_timeX
     t_thresholdType
     [ ?releaseMemory g_releaseMemory ]
     => o_digWave / n_digValue / nil
```

Description

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

Arguments

o_waveform	Analog input waveform.
n_vhi	High threshold value. This argument is used only when *t_thresholdType* is `hilo`.
n_vlo	Low threshold value. This argument is considered only when *t_thresholdType* is `hilo`.
n_vc	Central threshold value. This argument is used only when *t_thresholdType* is `center`.
n_timeX	The minimum duration of analog signal between *n_vhi* and *n_vlo*. It is used to determine the logic X.
t_thresholdType	

Type of logic threshold.

Valid values are:

- `hilo`: Any value greater than *`n_vhi`* is a high state (`1`). Any value less than *`n_vlo`* is a low state (`0`), and the rest are treated as unknown based on the value of *`n_timeX`*.
- `center`: Any value greater than *`n_vc`* is a high state (`1`). Any value less than *`n_vc`* is a low state (`0`).

`?releaseMemory` *`g_releaseMemory`*

Specifies whether to internally release the memory of the analog waveform after converting it into a digital waveform.

Valid values are:

- `t`: Memory of the analog waveform is released after its conversion into a digital waveform.
- `nil`: Memory of the analog waveform is not released even after its conversion into a digital waveform. This is the default value.

Value Returned

`o_digWave`	Waveform ID or a list of waveform IDs if the specified analog input is *`o_wave`*.
`n_digValue`	A scalar value if the specified analog input is `o_val`.
`nil`	Analog input cannot be converted into digital output because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
win=awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified results directory.

```
openResults("./simulation/lib/cell/maestro/results/maestro/ExplorerRun.0/1/
myTest/psf")
=> "./simulation/lib/cell/maestro/results/maestro/ExplorerRun.0/1/myTest/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
=>
tran(tranOp model instance output designParamVals
    primitives subckts variables
)
```

The following example selects the result `tran`.

```
selectResults('tran)
=> stdobj@0x2f5da0c8
```

The following example lists the outputs available in the selected result.

```
outputs()
=> ("/net1" "/net2")
```

The following example creates a waveform object `net1`, representing the waveform of the analog signal `net1`.

```
net1=v("net1")
=> srrWave:0x3548f020
```

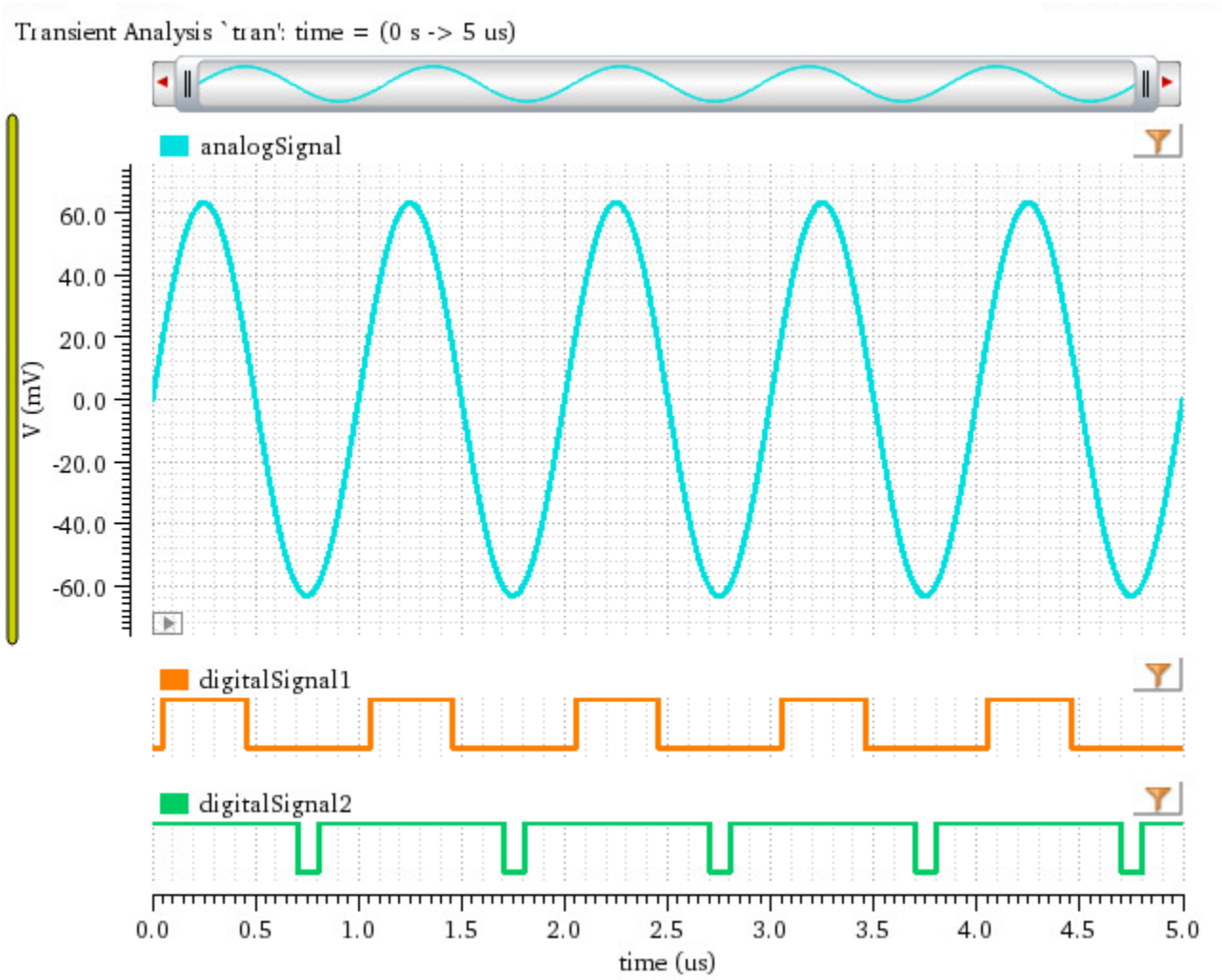
The following examples create two waveform objects, `dig1` and `dig2` that are generated after converting the analog signal `net1` into digital signals.

```
dig1=awvAnalog2Digital(net1 nil nil 20m nil "center")
=> srrWave:0x3548f030
dig2=awvAnalog2Digital(net1 -60m 60m nil 500n "hilo")
=> srrWave:0x3548f040
```

The following example plots the analog signal `net1` and digital signals `dig1` and `dig2` in the current Waveform window.

```
awvPlotWaveform(
                win
                list(net1 dig1 dig2)
                ?expr list("analogSignal" "digitalSignal1" "digitalSignal2")
                ?color list("y12" "y6" "y18")
                ?lineType list("line" "line" "line")
                ?lineStyle list("solid" "solid" "solid")
                ?lineThickness list("thick" "thick" "thick")
                )
```

```
=> t
```

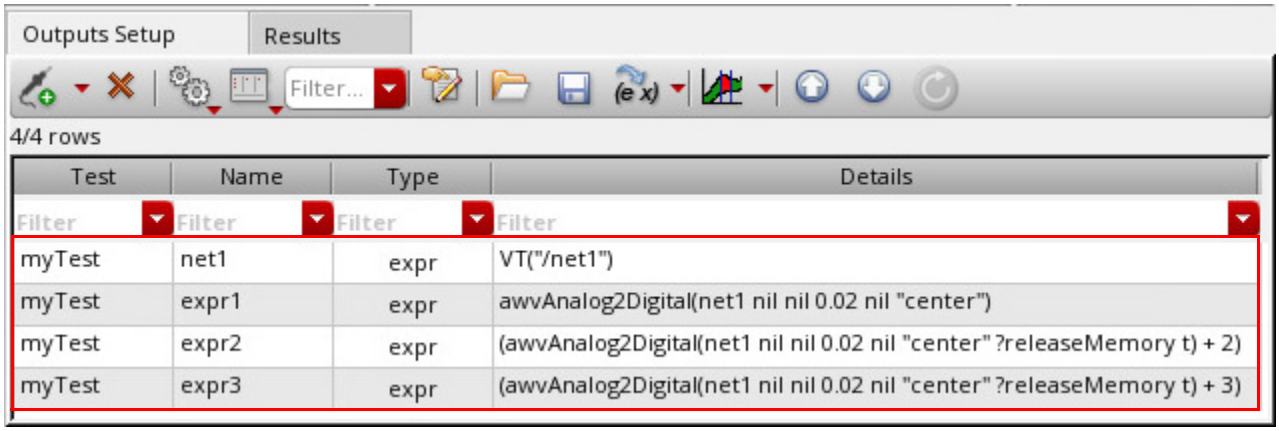


Evaluation Error when Using the releaseMemory Argument

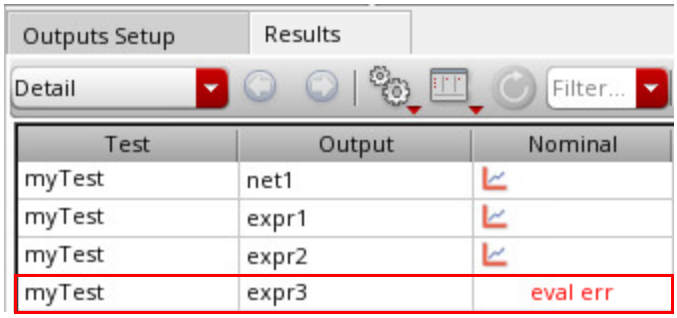
Consider an example where you have following expressions in the *Outputs Setup* tab of ADE Assembler.

Name	Expression
net1	`VT("/net1")`
expr1	`awvAnalog2Digital(net1 nil nil 0.02 nil "center")`
expr2	`(awvAnalog2Digital(net1 nil nil 0.02 nil "center" ?releaseMemory t) + 2)`

Name	Expression
expr3	(awvAnalog2Digital(net1 nil nil 0.02 nil "center" ?releaseMemory t) + 3)



When you run simulation, all expressions are evaluated successfully, except the expression `expr3`. This expression reports an evaluation error (*eval err*), as shown in the following figure.



Evaluation error

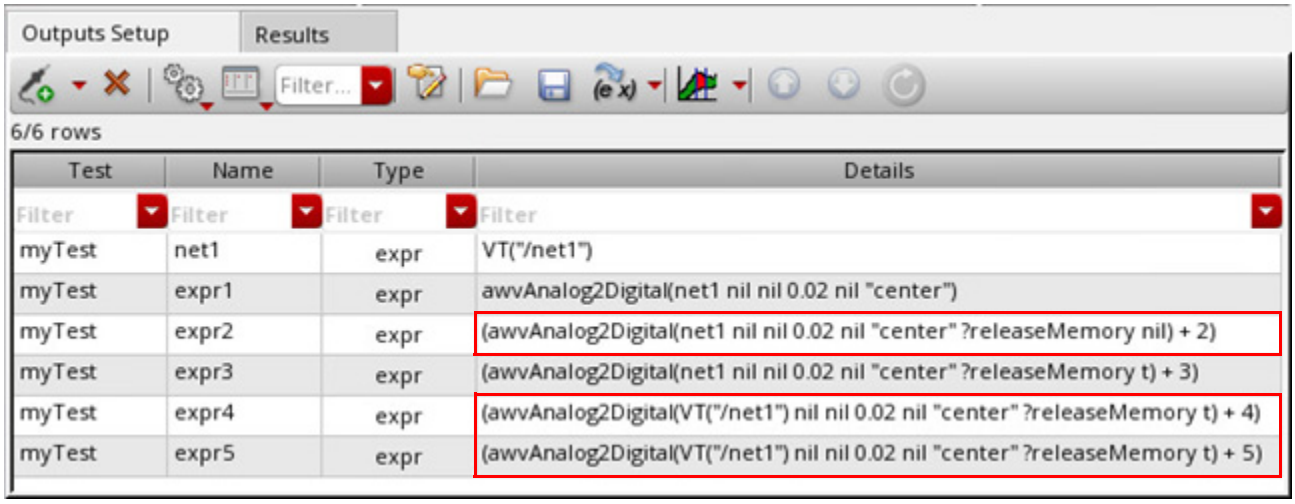
The first expression `net1` returns the transient waveform for the specified net `net1`. The analog waveform `net1` is stored in the cache and used further in evaluating expressions `expr1`, `expr2`, and `expr3`.

After evaluating the expression `expr2`, memory of the analog waveform `net1` is released because the argument `?releaseMemory` is set to `t`. Therefore, when the expression `expr3` is evaluated, an evaluation error (*eval err*) occurs.

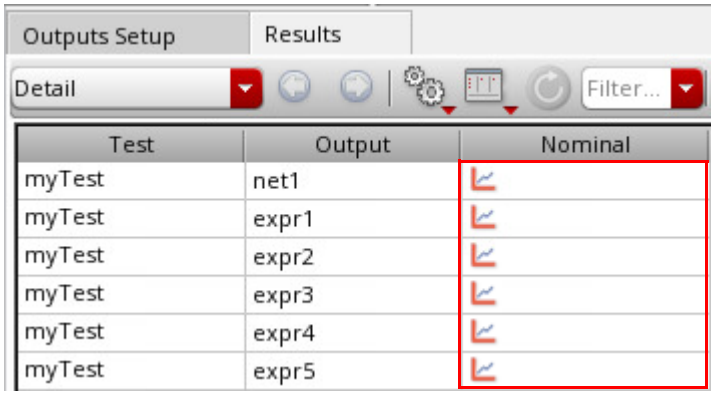
To resolve this error, you can either set the `?releaseMemory` argument to `nil` or use `VT("/net1")` instead of `net1` in these expressions:

```
(awvAnalog2Digital(net1 nil nil 0.02 nil "center" ?releaseMemory nil) + 2)
(awvAnalog2Digital(net1 nil nil 0.02 nil "center" ?releaseMemory t) + 3)
(awvAnalog2Digital(VT("/net1") nil nil 0.02 nil "center" ?releaseMemory t) + 4)
(awvAnalog2Digital(VT("/net1") nil nil 0.02 nil "center" ?releaseMemory t) + 5)
```

The following figures shows these expressions specified in the *Outputs Setup* tab of ADE Assembler.



The following figure shows that the specified expressions are successfully evaluated and their outputs are displayed in the *Results* tab of ADE Assembler.



Successful evaluation

awvAppendExpression

```
awvAppendExpression(
     w_windowID
     t_expr
     l_context
     [ ?expr l_exprList ]
     [ ?index l_waveIndexList ]
     [ ?color l_colorList ]
     [ ?lineType l_lineTypeList ]
     [ ?lineStyle l_lineStyleList ]
     [ ?lineThickness l_lineThicknessList ]
     [ ?showSymbols l_showList ]
     [ ?dataSymbol l_symbolList ]
     [ ?barBase t_barBase ]
     [ ?barWidth t_barWidth ]
     [ ?barShift t_barShift ]
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

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

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

Arguments

`w_windowID`	Waveform window ID.
`t_expr`	String containing an expression that is evaluated when the command is issued and reevaluated at the completion of each simulation in auto-update mode.
`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_exprList`*	List of waveform names to be displayed in the trace legend.
`?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 of colors for the waveforms. Available colors are defined in your technology file. Valid values are from `y1` through `y66`.

If you do not specify this argument, default colors are used.

`?lineType` *`l_lineTypeList`*

List specifying the type of line to be used for the waveforms.

Valid values are `line`, `bar`, `scatterPlot`, `poleZero`.

If you do not specify this argument, default value `line` is used.

`?lineStyle` *`l_lineStyleList`*

List specifying the line style to used for the waveforms.

Valid values are `solid`, `dash`, `dot`, `dashDot`, and `dashDotDot`.

If you do not specify this argument, default value `solid` is used.

`?lineThickness` *`l_lineThicknessList`*

List specifying the line thickness of the waveforms.

Valid values are `fine`, `medium`, `thick`, and `extraThick`.

If you do not specify this argument, default value `fine` is used.

`?showSymbols` *`l_showList`*

List of flags that specify whether to show the symbols on the waveforms. Valid values are `t` and `nil`.

The default value is `nil`, which means that symbols are not displayed on the waveforms.

The number of flags in the *`l_showList`* must match the number of symbols in the *`l_symbolList`*.

`?dataSymbol` *`l_symbolList`*

List of symbols to be displayed for data points on the waveforms.

To use a symbol, specify an integer or a single character corresponding to the symbol.

`?barBase` *`t_barBase`*

Base of the bar.

`?barWidth` *`t_barWidth`*

Width of the bar.

This argument takes integer values. The default value is `1`.

`?barShift` *`t_barShift`*

Specifies whether to shift the bar ahead or backwards when the `?lineType` argument is set to `bar`. This argument takes integer values.

Positive integer shifts the bar backwards and negative integer shifts the bar forward.

The default value is `0`, which indicates that bar is not shifted.

`?subwindow` *`x_subwindow`*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Expression *`t_expr`* is evaluated successfully and the resulting waveforms are added to Waveform window.
`nil`	Expression cannot be evaluated because of an error.

Examples

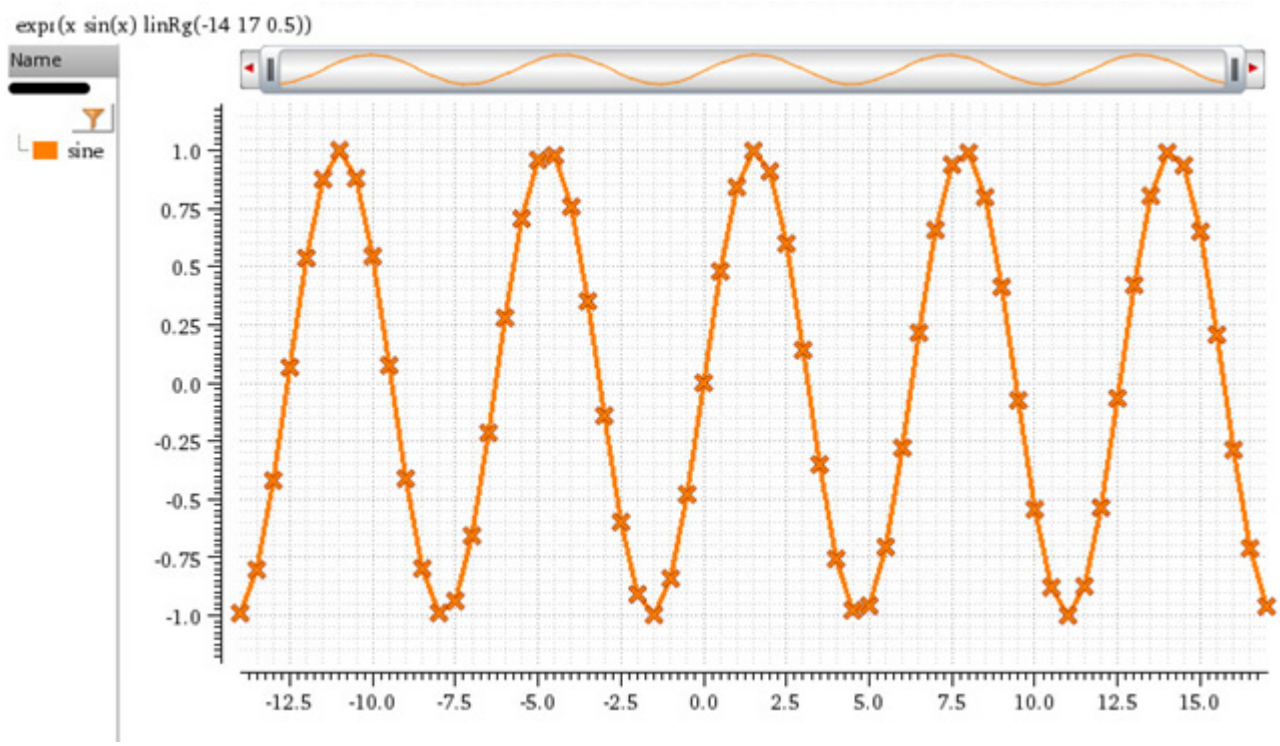
The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example evaluates the expression `sin(x)` from `x=-14.0` to `x=17`. The expression is evaluated at `x=-14.0`, `-13.5`, `-13.0`,...,`16.0`, `16.5`, and `17.0`. The resulting waveform is labeled `sine` in the Waveform window, and displayed in the `y6` layer color.

```
awvAppendExpression(
                    window(3)
```

```
                    "expr(x sin(x) linRg(-14 17 0.5))"
                    nil
                    ?expr list("sine")
                    ?color list("y6")
                    ?lineType list("line")
                    ?lineStyle list("solid")
                    ?lineThickness list("thick")
                    ?showSymbols list(t)
                    ?dataSymbol list("x")
                    )
=> t
```



Related Topics

Symbols for Data Points

awvAppendList

```
awvAppendList(
     w_windowID
     l_yListList
     l_xList
     [ ?expr l_exprList ]
     [ ?index l_waveIndexList ]
     [ ?color l_colorList ]
     [ ?lineType l_lineTypeList ]
     [ ?lineStyle l_lineStyleList ]
     [ ?lineThickness l_lineThicknessList ]
     [ ?showSymbols l_showList ]
     [ ?dataSymbol l_symbolList ]
     [ ?barBase t_barBase ]
     [ ?barWidth t_barWidth ]
     [ ?barShift t_barShift ]
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Plots y values in *l_yListList* against x values in *l_xListList* and adds the resulting waveforms to a subwindow.

The new waveforms are plotted in the same strip and y axis as their corresponding curve in *l_waveIndexList*. Also, the new waveforms are assigned the next lowest unassigned numbers. If *l_waveIndexList* contains curves 1 and 2, the curves resulting from plotting 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	A list of lists that specifies y-axis values. Each list must have the same number of items.
l_xList	A list that specifies x-axis values. Number of items in this list must be equal to the number of items in each list specified by *l_yListList*.

`?expr` *`l_exprList`*

List of waveform names to be displayed in the trace legend.

`?index` *`l_waveIndexList`*

List of integers identifying existing waveform curves that were plotted with one of the Waveform window plot functions.

If you do not specify index numbers for the waveforms in *`l_waveIndexList`*, the lowest unused numbers in the subwindow are assigned.

`?color` *`l_colorList`*

List of colors for the waveforms. Available colors are defined in your technology file. Valid values are from `y1` through `y66`.

If you do not specify this argument, default colors are used.

`?lineType` *`l_lineTypeList`*

List specifying the type of line to be used for the waveforms.

Valid values are `line`, `bar`, `scatterPlot`, `poleZero`.

If you do not specify this argument, default value `line` is used.

`?lineStyle` *`l_lineStyleList`*

List specifying the line style to used for the waveforms.

Valid values are `solid`, `dash`, `dot`, `dashDot`, and `dashDotDot`.

If you do not specify this argument, default value `solid` is used.

`?lineThickness` *`l_lineThicknessList`*

List specifying the line thickness of the waveforms.

Valid values are `fine`, `medium`, `thick`, and `extraThick`.

If you do not specify this argument, default value `fine` is used.

`?showSymbols` *`l_showList`*

List of flags that specify whether to show the symbols on the waveforms. Valid values are `t` and `nil`.

The default value is `nil`, which means that symbols are not displayed on the waveforms.

The number of flags in the *`l_showList`* must match the number of symbols in the *`l_symbolList`*.

`?dataSymbol` *`l_symbolList`*

List of symbols to be displayed for data points on the waveforms.

To use a symbol, specify an integer or a single character corresponding to the symbol.

`?barBase` *`t_barBase`*

Base of the bar.

`?barWidth` *`t_barWidth`*

Width of the bar.

This argument takes integer values. The default value is `1`.

`?barShift` *`t_barShift`*

Specifies whether to shift the bar ahead or backwards when the `?lineType` argument is set to `bar`. This argument takes integer values.

Positive integer shifts the bar backwards and negative integer shifts the bar forward.

The default value is `0`, which indicates that bar is not shifted.

`?subwindow` *`x_subwindow`*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

`?yNumber` *`l_yNumberList`*

List of integers identifying the y axis. Valid values are from `1` through `4`.

`?stripNumber` *`l_stripNumberList`*

List of integers identifying the strips.

Value Returned

`t`	Specified y-axis values are plotted against the specified x-axis values and the corresponding waveforms are displayed in the specified window.
`nil`	Waveforms cannot be plotted because of an error.

Examples

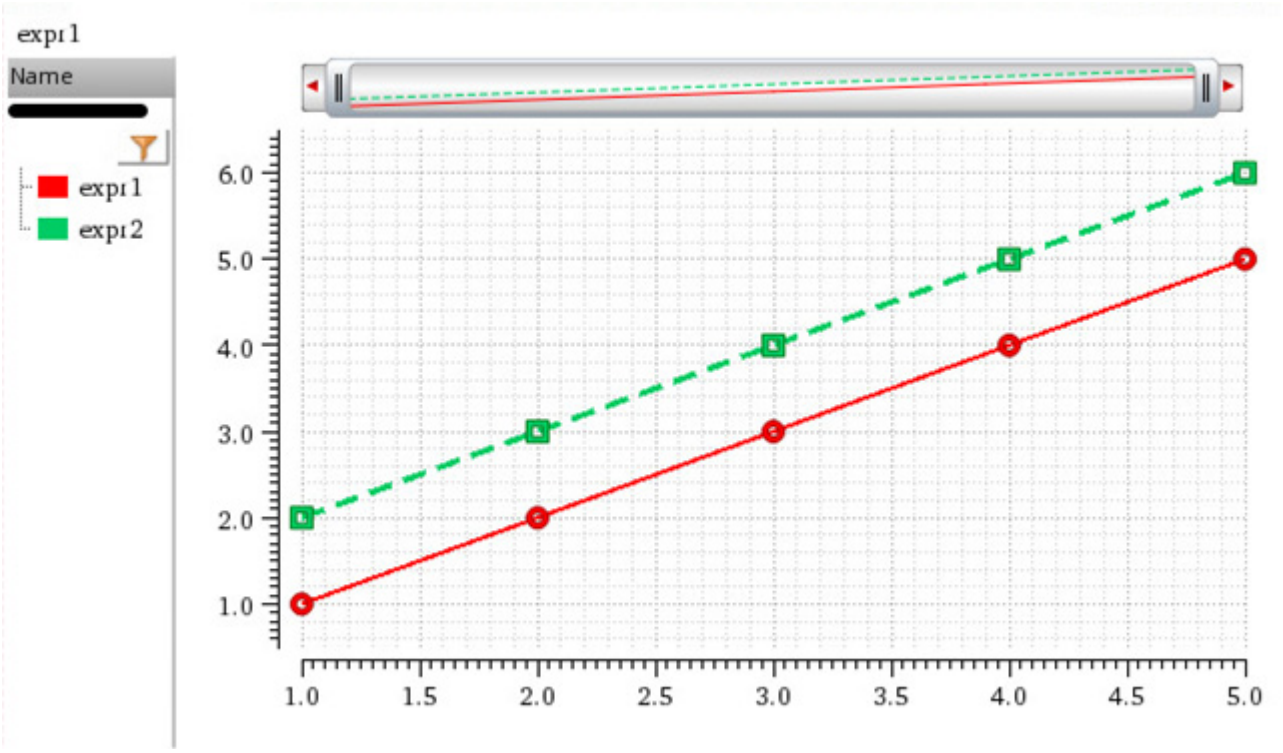
The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example plots y-axis values (`1`, `2`, `3`, `4`, `5`) and (`2`, `3`, `4`, `5`, `6`) against x-axis values (`1`, `2`, `3`, `4`, `5`) as waveforms with names `expr1` and `expr2`. The waveforms `expr1` and `expr2` are assigned with the index numbers `3` and `4`, respectively.

```
awvPlotList(window(3) list(list(1 2 3 4 5) list(2 3 4 5 6)) list(1 2 3 4 5) ?expr
list("expr1" "expr2") ?index list(3 4) ?color list("y1" "y66") ?lineType
list("line" "line") ?lineStyle list("solid" "dash") ?lineThickness list("medium"
"thick") ?showSymbols list(t t) ?dataSymbol list(4 5))
=> t
```

Note that `expr1` uses a symbol corresponding to integer `4`, which is a `circle`. `expr2` uses a symbol corresponding to integer `5`, which is a `square`.



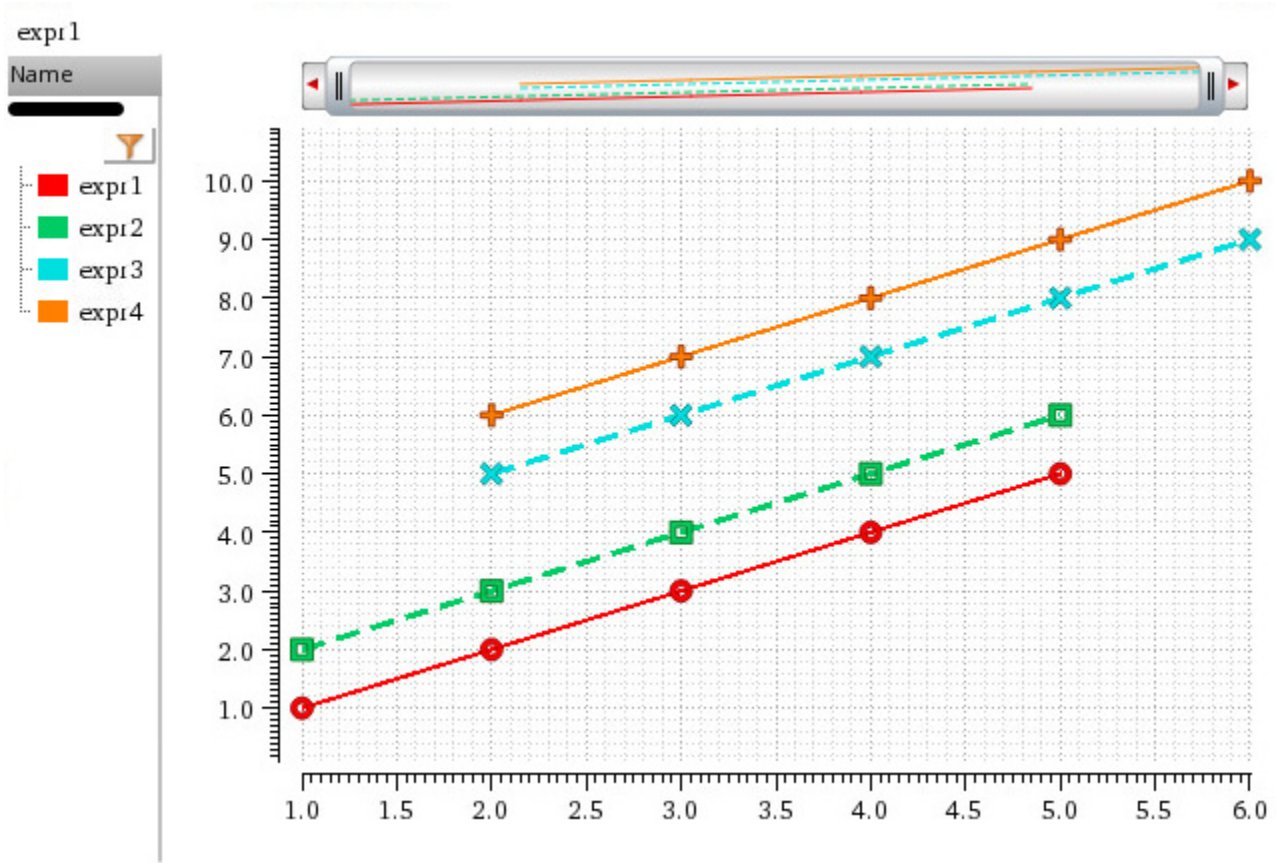
The following function returns the index numbers and names of the waveforms plotted in the window 3.

```
awvGetWaveNameList(window(3))
=>
((3 4)
    ("expr1" "expr2")
)
```

The following example plots y-axis values (`5`, `6`, `7`, `8`, `9`) and (`6`, `7`, `8`, `9`, `10`) against x-axis values (`2`, `3`, `4`, `5`, `6`) as waveforms with names `expr3` and `expr4` and appends them in the Waveform window with ID `3`.

```
awvAppendList(window(3) list(list(5 6 7 8 9) list(6 7 8 9 10)) list(2 3 4 5 6) ?expr
list("expr3" "expr4") ?color list("y12" "y6") ?lineType list("line" "line")
?lineStyle list("dash" "solid") ?lineThickness list("thick" "medium") ?showSymbols
list(t t) ?dataSymbol list("x" "+"))
=> t
```

Note that `expr3` uses a symbol corresponding to character `"x"`, which is a `cross` sign. `expr4` uses a symbol corresponding to character `"+"`, which is a `plus` sign.



The following example returns the index numbers and names of waveforms plotted in the Waveform window with ID `3`.

```
awvGetWaveNameList(window(3))
=>
((3 4 5 6)
    ("expr1" "expr2" "expr3" "expr4")
)
```

Related Topics

Symbols for Data Points

awvAppendWaveform

```
awvAppendWaveform(
     w_windowID
     l_waveformList
     [ ?subwindow x_subwindow ]
     [ ?expr l_exprList ]
     [ ?index l_waveIndexList ]
     [ ?component t_component ]
     [ ?color l_colorList ]
     [ ?lineType l_lineTypeList ]
     [ ?lineStyle l_lineStyleList ]
     [ ?lineThickness l_lineThicknessList ]
     [ ?showSymbols l_showList ]
     [ ?dataSymbol l_symbolList ]
     [ ?barBase t_barBase ]
     [ ?barWidth t_barWidth ]
     [ ?barShift t_barShift ]
     )
     => t / nil
```

Description

Plots the waveforms in the list *l_waveformList* and appends them in a subwindow of the specified Waveform window.

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_waveformList	A list of waveform objects to be plotted.

?subwindow *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, waveforms are plotted in the current subwindow of the specified Waveform window.

`?expr` *`l_exprList`*

List of waveform names to be displayed in the trace legend.

`?index` *`l_waveIndexList`*

List of index numbers identifying the waveforms.

If you do not specify index numbers for the waveforms in *`l_waveIndexList`*, the lowest unused numbers in the subwindow are assigned.

`?color` *`l_colorList`*

List of colors for the waveforms. Available colors are defined in your technology file. Valid values are from `y1` through `y66`.

If you do not specify this argument, default colors are used.

The color information can also be provided in the RGB format, such as `0X00FF50`.

`?lineType` *`l_lineTypeList`*

List specifying the type of line to be used for the waveforms.

Valid values are `line`, `bar`, `scatterPlot`, `poleZero`.

If you do not specify this argument, default value `line` is used.

`?lineStyle` *`l_lineStyleList`*

List specifying the line style to used for the waveforms.

Valid values are `solid`, `dash`, `dot`, `dashDot`, and `dashDotDot`.

If you do not specify this argument, default value `solid` is used.

`?lineThickness` *`l_lineThicknessList`*

List specifying the line thickness of the waveforms.

Valid values are `fine`, `medium`, `thick`, and `extraThick`.

If you do not specify this argument, default value `fine` is used.

`?showSymbols` *`l_showList`*

List of flags that specify whether to show the symbols on the waveforms. Valid values are `t` and `nil`.

The default value is `nil`, which means that symbols are not displayed on the waveforms.

The number of flags in the *`l_showList`* must match the number of symbols in the *`l_symbolList`*.

`?dataSymbol` *`l_symbolList`*

List of symbols to be displayed for data points on the waveforms.

To use a symbol, specify an integer or a single character corresponding to the symbol.

`?barBase` *`t_barBase`*

Specifies base of the bar when the `?lineType` argument is set to `bar`.

`?barWidth` *`t_barWidth`*

Specifies width of the bar when the `?lineType` argument is set to `bar`.

This argument takes integer values. The default value is `1`.

`?barShift` *`t_barShift`*

Specifies whether to shift the bar ahead or backwards when the `?lineType` argument is set to `bar`. This argument takes integer values.

Positive integer shifts the bar backwards and negative integer shifts the bar forward.

The default value is `0`, which indicates that bar is not shifted.

Value Returned

`t`	Waveforms specified in the *`l_waveformList`* list are plotted in the specified Waveform window.
`nil`	Waveforms cannot be plotted because of an error.

Examples

The following example creates a Waveform window and returns its ID.

```
awvCreatePlotWindow()
=> window:3
```

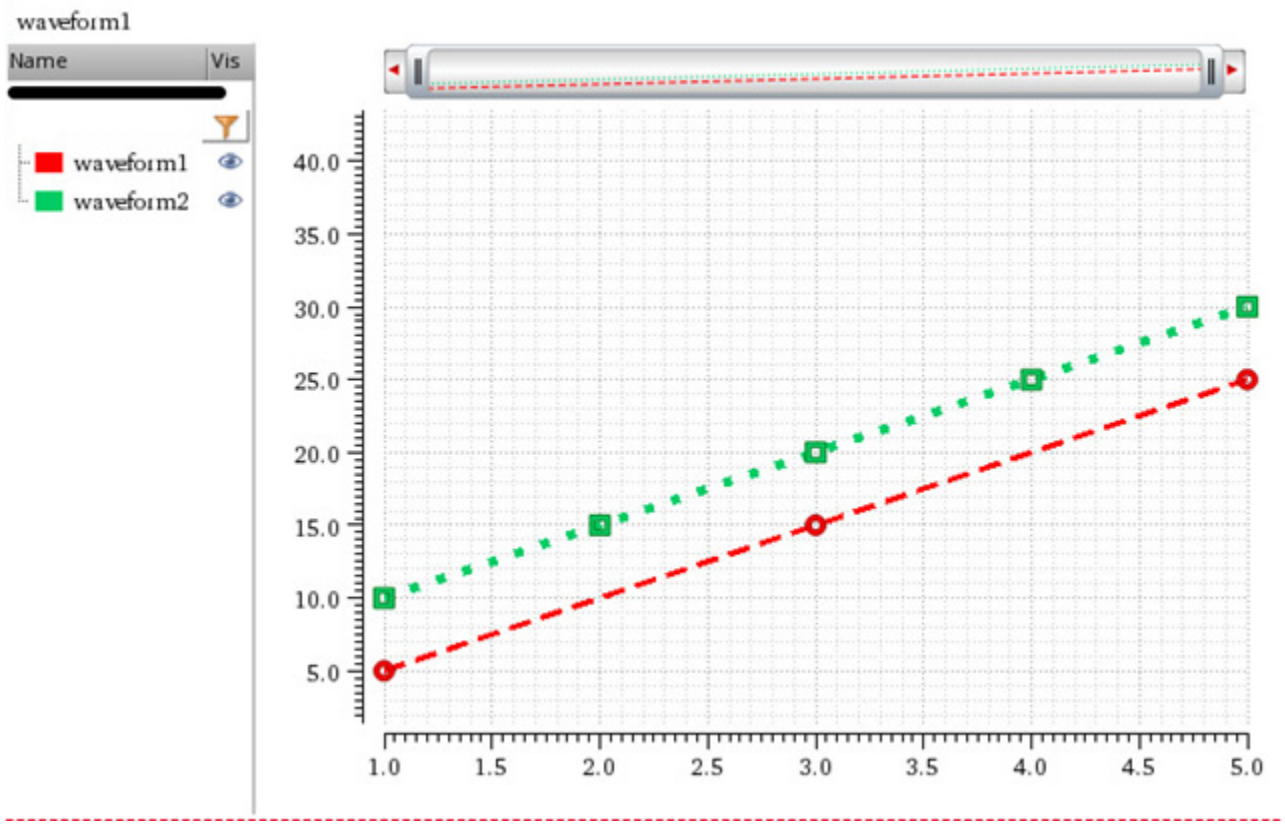
The following examples create waveform objects `w1`, `w2`, `w3`, and `w4`.

```
w1=drCreateWaveform(drCreateVec('double list(1 3 5)) drCreateVec('double list(5 15
25)))
=> srrWave:0x2d4a9020
w2=drCreateWaveform(drCreateVec('double list(1 2 3 4 5)) drCreateVec('double
list(10 15 20 25 30)))
=> srrWave:0x2d4a9030
w3=drCreateWaveform(drCreateVec('double list(1 3 5)) drCreateVec('double list(15
25 35)))
=> srrWave:0x2d4a9040
w4=drCreateWaveform(drCreateVec('double list(1 2 3 4 5)) drCreateVec('double
list(20 25 30 35 40)))
=> srrWave:0x2d4a9050
```

The following example plots the waveforms objects `w1` and `w2` specified in the list.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(w1 w2)
                ?expr list("waveform1" "waveform2")
                ?color list("y1" "y66")
                ?index list(1 2)
                ?lineType list("line" "line")
                ?lineStyle list("dash" "dot")
                ?lineThickness list("thick" "extraThick")
                ?showSymbols list(t t)
                ?dataSymbol list(4 5)
                )
```

```
=> t
```



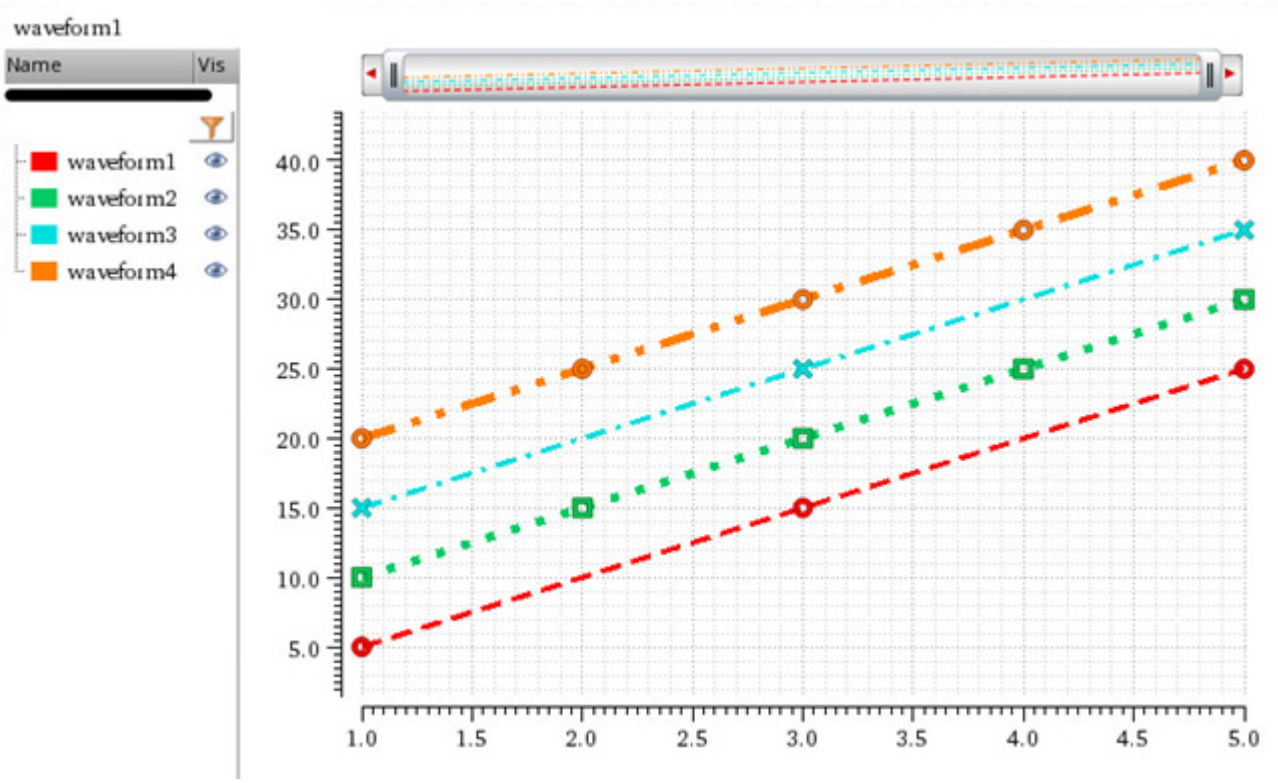
The following example returns the index numbers and names of the waveforms plotted using `awvPlotWaveform` function.

```
awvGetWaveNameList(window(3))
=>
((1 2)
    ("waveform1" "waveform2")
)
```

The following example plots the waveform objects `w3` and `w4` specified in the list and appends them to the waveforms plotted using the `awvPlotWaveform` function.

```
awvAppendWaveform(
                    awvGetCurrentWindow()
                    list(w3 w4)
                    ?expr list("waveform3" "waveform4")
                    ?color list("y12" "y6")
                    ?lineType list("line" "line")
                    ?lineStyle list("dashDot" "dashDotDot")
                    ?lineThickness list("thick" "extraThick")
```

```
                    ?showSymbols list(t t)
                    ?dataSymbol list("x" "O")
                    )
=> t
```



The following example returns the index numbers and names of the waveforms plotted in the specified Waveform window.

```
awvGetWaveNameList(window(3))
=>
((1 2 3 4)
    ("waveform1" "waveform2" "waveform3" "waveform4")
)
```

awvClearSubwindowHistory

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

Description

Erases the contents from a subwindow of the specified Waveform window.

This function deletes the waveforms, title, date stamp, and labels stored in the internal memory. The other subwindows are not affected.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow (found in the top-right corner of the subwindow) whose contents are to be erased. If you do not specify this argument, contents of the current subwindow are erased.

Value Returned

t	Contents of the subwindow are erased.
nil	Contents of the subwindow cannot be erased because the specified Waveform window or the subwindow does not exist.

Examples

The following example erases contents of the subwindow 3 from the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:4
awvClearSubwindowHistory(win ?subwindow 3)
=> t
```

awvClearPlotWindow

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

Description

Clears the graphs shown in the specified Waveform window. History for the window and the subwindows are maintained.

Arguments

w_windowID	Waveform window ID.

Value Returned

t	Identification number of the subwindow, which can be found in the top-right corner of the subwindow.
nil	The subwindow cannot be added because the specified Waveform window does not exist.

Examples

The following example clears the graphs from the current Waveform window.

```
win=awvGetCurrentWindow()
=> window:5
awvClearPlotWindow(win)
=> t
```

awvClearWindowHistory

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

Description

Erases the contents of the specified Waveform window.

This function deletes the waveforms, title, date stamp, and labels stored in the internal memory. By default, this function operates only on the subwindows whose update statuses are turned on.

Arguments

w_windowID	Waveform window ID.
?force *g_force*	Specifies whether the function operates on all subwindows of the specified Waveform window. Valid values are: ■ `t`: The functions operates on all subwindows of the Waveform window. ■ `nil`: The function operates only on the subwindows whose update statuses are turned on. This is the default value.

Value Returned

`t`	Contents of the Waveform window are erased.
`nil`	Contents of the Waveform window cannot be erased because the specified Waveform window does not exist.

Examples

The following example erases contents from all subwindows of the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:4
```

```
awvClearWindowHistory(win ?force t)
=> t
```

The following example erases contents only from the subwindows whose update statuses are turned on in the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:4
awvClearWindowHistory(win)
=> t
```

awvCloseCalculator

```
awvCloseCalculator(
     [ t_sessionName ]
     )
     => t / nil
```

Description

Closes the Calculator invoked from the current session or from the specified session.

Arguments

t_sessionName	Name of the session. This is an optional argument. If you do not specify this argument, the current session is used.

Value Returned

t	The Calculator is closed.
nil	The Calculator cannot be closed because the specified session does not exist.

Examples

The following example closes the Calculator from the current session.

```
awvCloseCalculator()
=> t
```

The following example closes the Calculator invoked from the specified session.

```
session=maeGetSessions()
=> ("fnxSession1")
awvCloseCalculator("fnxSession1")
=> t
```

awvCloseWindow

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

Description

Closes the specified Waveform window.

Arguments

w_windowID	Waveform window ID.

Value Returned

t	Waveform window is closed.
nil	The specified Waveform window does not exist.

Examples

The following example closes the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:4
awvCloseWindow(win)
=> t
```

awvCloseWindowMenuCB

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

Description

Closes the current window. This function is defined in `dfII/etc/context/awv.cxt`.

Arguments

None

Value Returned

`t`	Windows are closed.
`nil`	There are no windows currently open.

Examples

The following example closes windows that are currently open.

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

awvCreateBus

```
awvCreateBus(
     t_busName
     l_waveList
     r_radix
     )
     => o_bus / nil
```

Description

Creates a digital bus from the specified digital waveforms and the radix type.

Arguments

t_busName	Name of the digital bus.
l_waveList	List of digital waveforms or expressions.
r_radix	Radix of the bus. Valid values are `Binary`, `Octal`, `Ascii`, `Hex`, `Signed Decimal`, and `Unsigned Decimal`.

Value Returned

o_bus	Waveform ID of the digital bus.
`nil`	Digital bus cannot be created because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/mixed_signal.raw")
"/servers/user/design/mixed_signal.raw"
```

The following example lists the results available in the currently open results directory.

```
results()
=>
```

```
(tranOp model instance output tran
    variables
)
```

The following example selects the `tran` result of the results directory.

```
selectResult('tran)
=> stdobj@0x37f03410
```

The following examples create digital waveform objects `dig1`, `dig2`, `dig3`, and `dig4` from the digital signals `net021`, `net027`, `net031`, and `net034` signals stored in the `tran` results of the results directory.

```
dig1=v("topLevel.net021")
=> srrWave:0x382bf020
dig2=v("topLevel.net027")
=> srrWave:0x382bf030
dig3=v("topLevel.net031")
=> srrWave:0x382bf040
dig4=v("topLevel.net034")
=> srrWave:0x382bf050
```

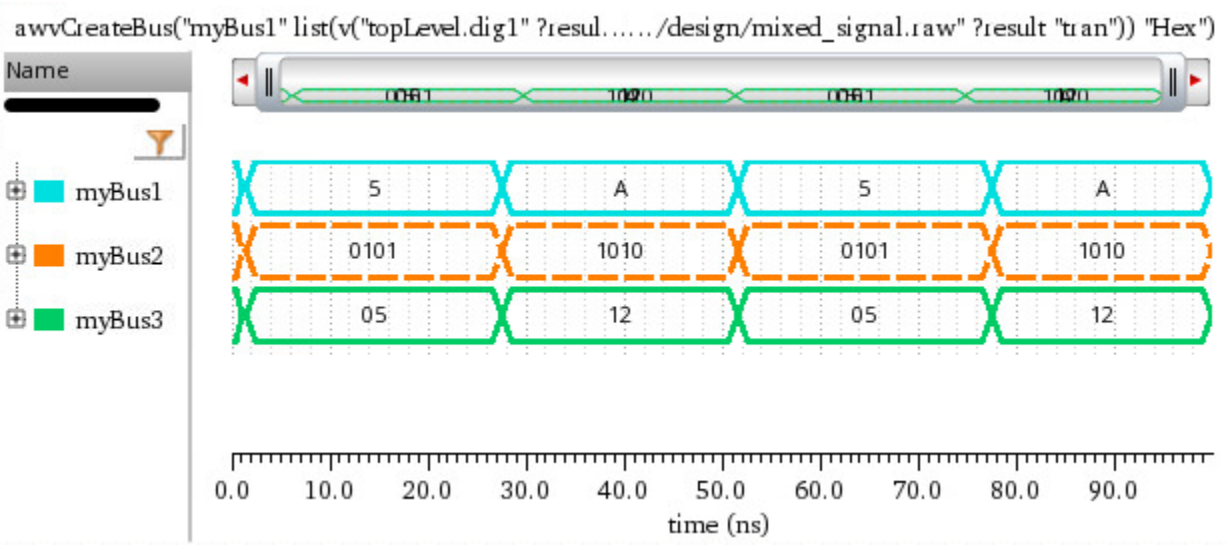
The following examples create waveform objects `myBus1`, `myBus2`, and `myBus3`, representing digital buses generated from the specified list of digital signals, `dig1`, `dig2`, `dig3`, and `dig4` and with the specified radix.

```
myBus1=awvCreateBus("myBus1" list(dig1 dig2 dig3 dig4) "Hex")
=> srrWave:0x382bf060
myBus2=awvCreateBus("myBus2" list(dig1 dig2 dig3 dig4) "Binary")
=> srrWave:0x382bf070
myBus3=awvCreateBus("myBus3" list(dig1 dig2 dig3 dig4) "Octal")
=> srrWave:0x382bf080
```

The following example plots the digital buses in the current Waveform window.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(myBus1 myBus2 myBus3)
                ?color list("y12" "y6" "y18")
                ?lineStyle list("solid" "dash" "solid")
                ?lineThickness list("thick" "thick" "thick")
                )
```

```
=> t
```



The following example creates a digital bus with the digital waveforms `dig1` and `dig2` with the `Binary` radix.

```
dig1= getData("topLevel.net021" ?result "tran" ?resultsDir "./mixed_signal.raw")
=> srrWave:0x397fa030
dig2= getData("topLevel.net034" ?result "tran" ?resultsDir "./mixed_signal.raw")
=> srrWave:0x397fa040
awvCreateBus("myBus" list(dig1 dig2) "Binary")
=> srrWave:0x397fa130
```

awvCreateBusFromWaveList

```
awvCreateBusFromWaveList(
     l_waveList
     [ ?busType t_busType ]
     )
     => o_bus / nil
```

Description

Creates a digital bus from a list of specified digital waveforms.

Arguments

l_waveList	List of digital waveforms or expressions.
?busType *t_busType*	
	Radix of the bus. Valid values are `Hex`, `Dec`, `Oct` and `Bin`. The default value is `Hex`.

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`	Digital bus cannot be created because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/mixed_signal.raw")
=> "/servers/user/design/mixed_signal.raw"
```

The following example lists the results available in the currently open results directory.

```
results()
```

```
=>
(tranOp model instance output tran
    variables
)
```

The following example selects the result `tran`.

```
selectResult('tran)
=> stdobj@0x347b0218
```

The following example creates digital waveform objects `dig1`, `dig2`, `dig3`, and `dig4` from the digital signals `net021`, `net027`, `net031`, and `net034` signals stored in the `tran` results of the results directory.

```
dig1=v("topLevel.net021")
=> srrWave:0x36228020

dig2=v("topLevel.net027")
=> srrWave:0x36228030

dig3=v("topLevel.net031")
=> srrWave:0x36228040

dig4=v("topLevel.net034")
=> srrWave:0x36228050
```

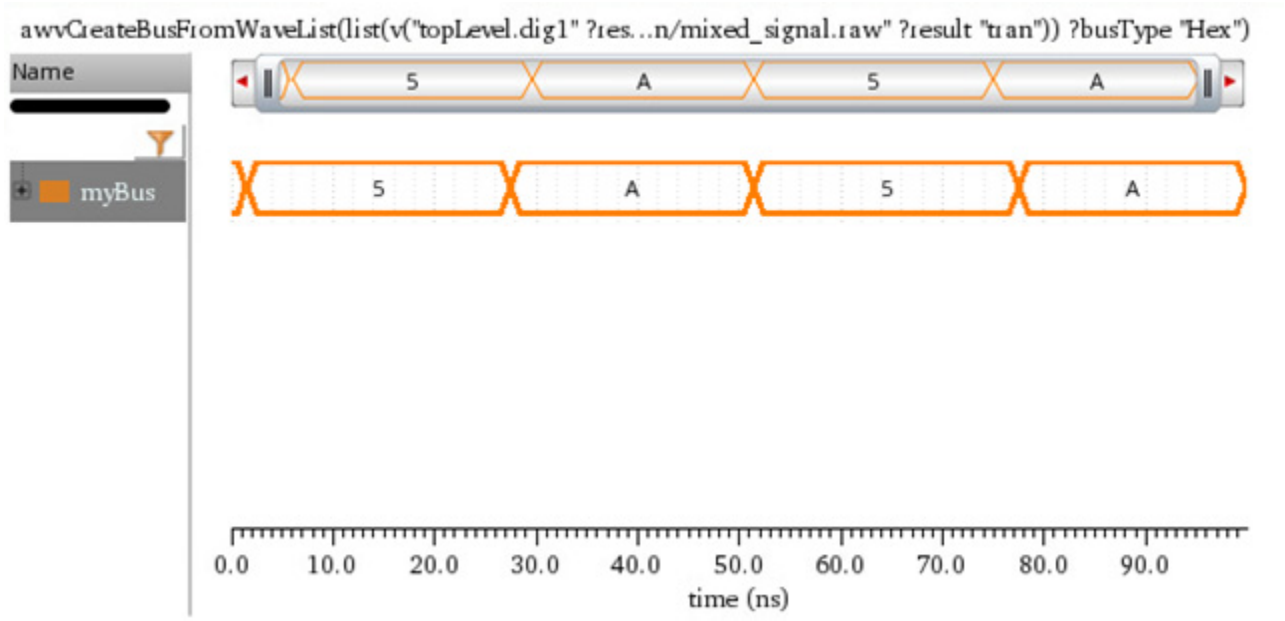
The following example creates a waveform object `busFromList`, representing a digital bus generated from the specified list of digital signals, `dig1`, `dig2`, `dig3`, and `dig4`.

```
busFromList=awvCreateBusFromWaveList(list(dig1 dig2 dig3 dig4))
=> srrWave:0x362280a0
```

The following example plots the digital bus in the current Waveform window.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(busFromList)
                ?expr list("myBus")
                ?color list("y6")
                )
```

```
=> t
```



The following example creates a digital bus with the digital waveforms `dig1`, `dig2`, and `dig3`.

```
dig1= getData("topLevel.net021" ?result "tran" ?resultsDir "./mixed_signal.raw")
=> srrWave:0x397fa1e0
dig2= getData("topLevel.net034" ?result "tran" ?resultsDir "./mixed_signal.raw")
=> srrWave:0x397fa1f0
dig3= getData("topLevel.dig1" ?result "tran" ?resultsDir "./mixed_signal.raw")
=> srrWave:0x397fa200
awvCreateBusFromWaveList(list(dig1 dig2 dig3))
=> srrWave:0x397fa250
```

awvCreatePlotWindow

```
awvCreatePlotWindow(
     [ ?parentWindow w_windowID ]
     [ ?graphType t_graphType ]
     )
     => w_windowID / nil
```

Description

Creates a Waveform window and returns its 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 technology file of the parent widow.

`?graphType` *`t_graphType`*

Type of the graph.

Valid values are `Rectangular`, `Tabular`, `Polar`, `Impedance`, `Admittance`, and `Immittance`.

The default graph type is `Rectangular`.

Value Returned

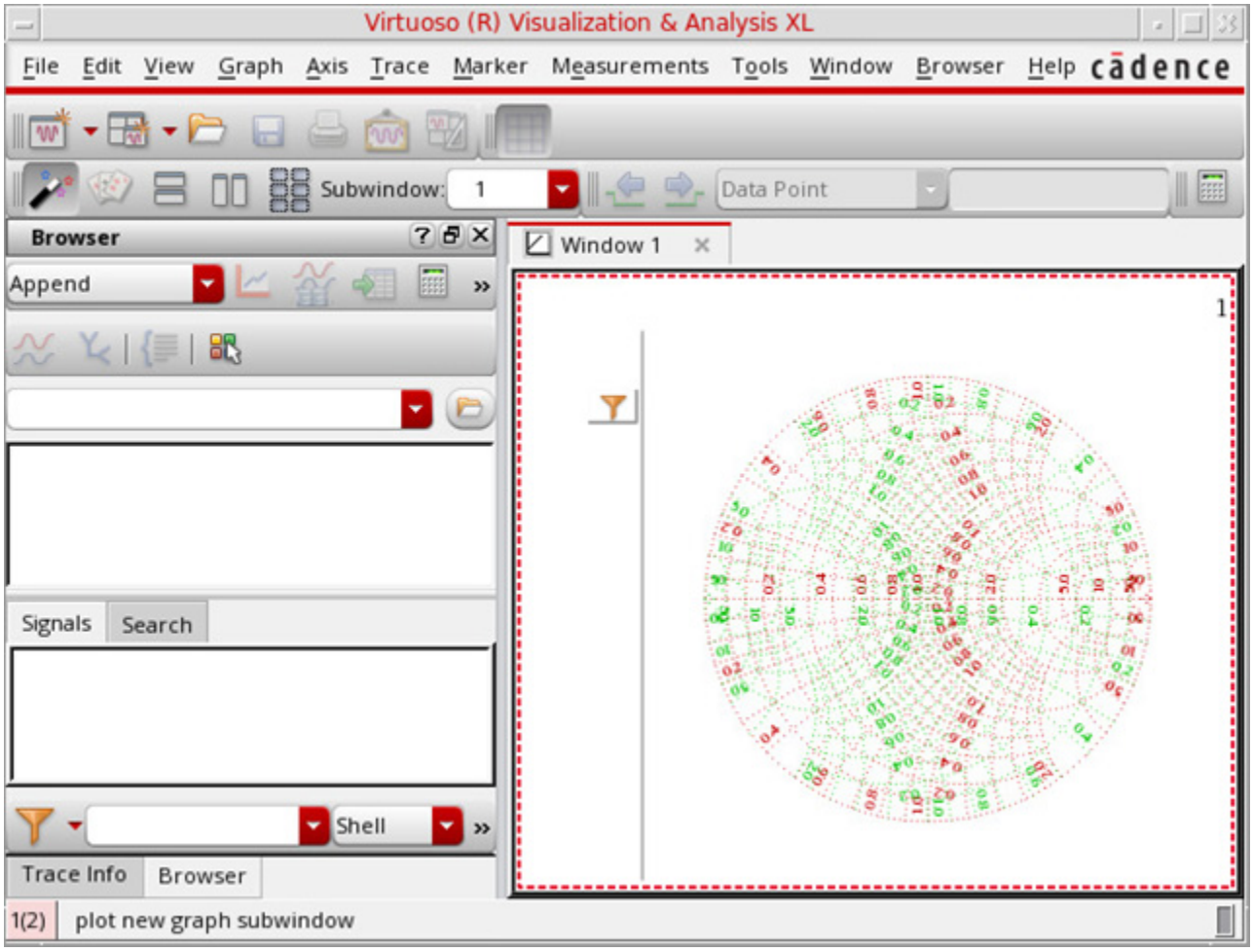
`w_windowID`	ID of the new Waveform window.
`nil`	Waveform window cannot be created because of an error.

Examples

The following example creates a Waveform window of `Immittance` graph type and returns its window ID.

```
awvCreatePlotWindow(?graphType "Immittance")
```

```
=> window:3
```



awvDeleteAllWaveforms

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

Description

Deletes all waveforms from a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow (found in the top-right corner of the subwindow) from which waveforms are to be deleted. If you do not specify this argument, waveforms are deleted from the current subwindow.

Value Returned

t	All waveforms are deleted from the subwindow.
nil	Waveforms cannot be deleted because the specified Waveform window or the subwindow does not exist.

Examples

The following example deletes all waveforms from the subwindow 2 of the current Waveform window.

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

The following example deletes all waveforms from the current subwindow of the specified Waveform window.

```
awvDeleteAllWaveforms(window(5))
=> t
```

awvDeleteMarker

```
awvDeleteMarker(
     w_windowID
     t_bookmarkIDs
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Deletes the specified bookmarks from a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
t_bookmarkIDs	ID of the bookmark to be deleted. To specify more than one bookmark IDs, use the following notation: `'(bm1 bm2 bm3)` Where bm1, bm2, and bm3 are the IDs of the bookmarks to be deleted.
`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Bookmarks are deleted successfully.
`nil`	The specified Waveform window, subwindow, or the bookmark ID does not exist.

Examples

The following example creates an `xrange` bookmark at the specified location in the current subwindow of the specified Waveform window.

```
bm1=awvPlaceBookmark(window(3) "xrange" list(50ns 300ns) ?waveIndex 1 ?description
"xRangeBookmark")
=> ("xrangeBookmark[1.1.1]")
```

The following example creates a `yrange` bookmark at the specified location in the current subwindow of the specified Waveform window.

```
bm2=awvPlaceBookmark(window(3) "yrange" list(-2.5 2.5) ?waveIndex 1 ?description
"yRangeBookmark")
=> ("yrangeBookmark[1.1.1]")
```

The following example deletes the xrange bookmark `bm1` from the current subwindow of the specified Waveform window.

```
awvDeleteMarker(window(3) bm1)
=> t
```

The following example deletes the yrange bookmark `bm2` from the current subwindow of the specified Waveform window.

```
awvDeleteMarker(window(3) bm2)
=> t
```

If you want to delete both bookmarks `bm1` and `bm2` together, use the following command:

```
awvDeleteMarker(window(3) '(bm1 bm2))
=> t
```

awvDeleteSubwindow

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

Description

Deletes a subwindow from the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow (found in the top-right corner of the subwindow) to be deleted. If you do not specify this argument, the current subwindow is deleted.

Value Returned

`t`	Subwindow is deleted.
`nil`	Subwindow cannot be deleted because the specified Waveform window or the subwindow does not exist.

Examples

The following example deletes the subwindow `4` from the current Waveform window.

```
win = awvGetCurrentWindow()
awvDeleteSubwindow(win ?subwindow 4)
=> t
```

awvDeleteWaveform

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

Description

Deletes the specified waveform from a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
x_index	ID of the waveform to be deleted. It is an integer value.
x_subwindow	Identification number of the subwindow (found in the top-right corner of the subwindow) from which the specified waveform is to be deleted. If you do not specify this argument, the specified waveform, if exists, is deleted from the current subwindow.

Value Returned

`t`	Waveform is deleted from the subwindow.
`nil`	Waveform cannot be deleted because the specified Waveform window, subwindow, or the waveform ID does not exist.

Examples

The following example deletes the waveform with the ID `5` from the subwindow `3` of the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:4
awvDeleteWaveform(win 5 ?subwindow 3)
=> t
```

The following example deletes the waveform with the ID `8` from the current subwindow of the specified Waveform window.

```
awvDeleteWaveform(window(89) 8)
=> t
```

awvDigital2Analog

```
awvDigital2Analog(
     o_waveform
     n_vHi
     n_vLo
     s_VX
     [ ?mode s_mode ]
     [ ?outWaveType s_outWaveType ]
     [ ?vprevSTART s_vprevSTART ]
     )
     => o_waveform / nil
```

Description

Returns the analog form of the specified digital waveform.

Arguments

o_waveform	Represents the digital waveform 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.
n_vHi	The high analog value to which the digital `1` (for single-bit waveform) or maximum possible bus value is converted.
n_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 *o_waveform* is a bus or it is a list with at least one bus. Valid values are: ■ `wavelist`: The input bus is converted into a list of analog waves, each representing a single-bit digital waveform in the bus. ■ `busvalue`: The output is a single analog wave, representing the value of the input bus.
?outWaveType *s_outWaveType*	

A string to be provided if *o_waveform* is a bus or it is a list with at least one bus.

Valid values are:

- pwl: The points in the output analog waveform are joined by straight line segments.
- zeroT: The output analog waveform has voltage transitions in zero time.

?vprevSTART *s_vprevSTART*

The initial value of vprev to be used.

This is required if the specified digital waveform starts at state X and vprev is used in the expression for 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 is a single-bit digital waveform or if the input is a bus and the specified *s_mode* is busvalue. A list of waveforms is returned if the input is a list of single-bit digital waveforms or if the input is a bus and the specified *s_mode* is wavelist.
nil	Analog output of the specified digital input cannot be returned because of an error.

Examples

The following example returns the waveform of the signal topLevel.net[0].

```
w1=v("topLevel.net[0]" ?result "tran" ?resultsDir "/servers/user/design/
mixed_signal.raw")
=> srrWave:0x3603d1b0
```

The following example returns the waveform of the signal topLevel.net[1].

```
w2=v("topLevel.net[1]" ?result "tran" ?resultsDir "/servers/user/design/
mixed_signal.raw")
=> srrWave:0x3603d1c0
```

The following example converts the analog waveform `w1` to a digital waveform `w3`.

```
w3=awvAnalog2Digital(w1 1.8 0 nil 20n  "hilo")
=> srrWave:0x3603d1e0
```

The following example converts the analog waveform `w2` to a digital waveform `w4`.

```
w4=awvAnalog2Digital(w2 1.7 0 nil 20n  "hilo")
=> srrWave:0x3603d1f0
```

The following example creates a bus `busSignal` from two digital signals `w3` and `w4`.

```
busSignal=awvCreateBus("myBus" list(w3 w4) "Binary")
=> srrWave:0x3603d200
```

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

```
w5=awvDigital2Analog(busSignal 1.8 0 "vhi+vlo/2.0" ?mode "busvalue" ?outWaveType
"zeroT")
=> srrWave:0x3603d210
```

Returns a list of analog waveforms.

```
waveList=awvDigital2Analog(busSignal 1.8 0 "vhi+vlo/2.0" ?mode "wavelist"
?outWaveType "pwl")
=> (srrWave:0x3603d230 srrWave:0x3603d240)
```

The following examples create waveform objects `wave0`, `wave1`, `wave2`, and `wave3` with the specified x- and y-vector values.

```
list0=list(list(0.0 1e-06 1e-06 3e-06) list(0.0 0.0 5.0 5.0))
    xvec=drCreateVec('double car(list0))
    yvec=drCreateVec('double cadr(list0))
    wave0=drCreateWaveform(xvec yvec)
    drGetWaveformXVec(wave0)~>units="s"
    drGetWaveformXVec(wave0)~>name="time"
=>
((0.0 1e-06 1e-06 3e-06)
    (0.0 0.0 5.0 5.0)
)
=> srrVec:0x2d740020
=> srrVec:0x2d740030
=> srrWave:0x2d784020
=> "s"
=> "time"

list1=list(list(0.0 2e-06 2e-06 3e-06) list(0.0 0.0 5.0 5.0))
```

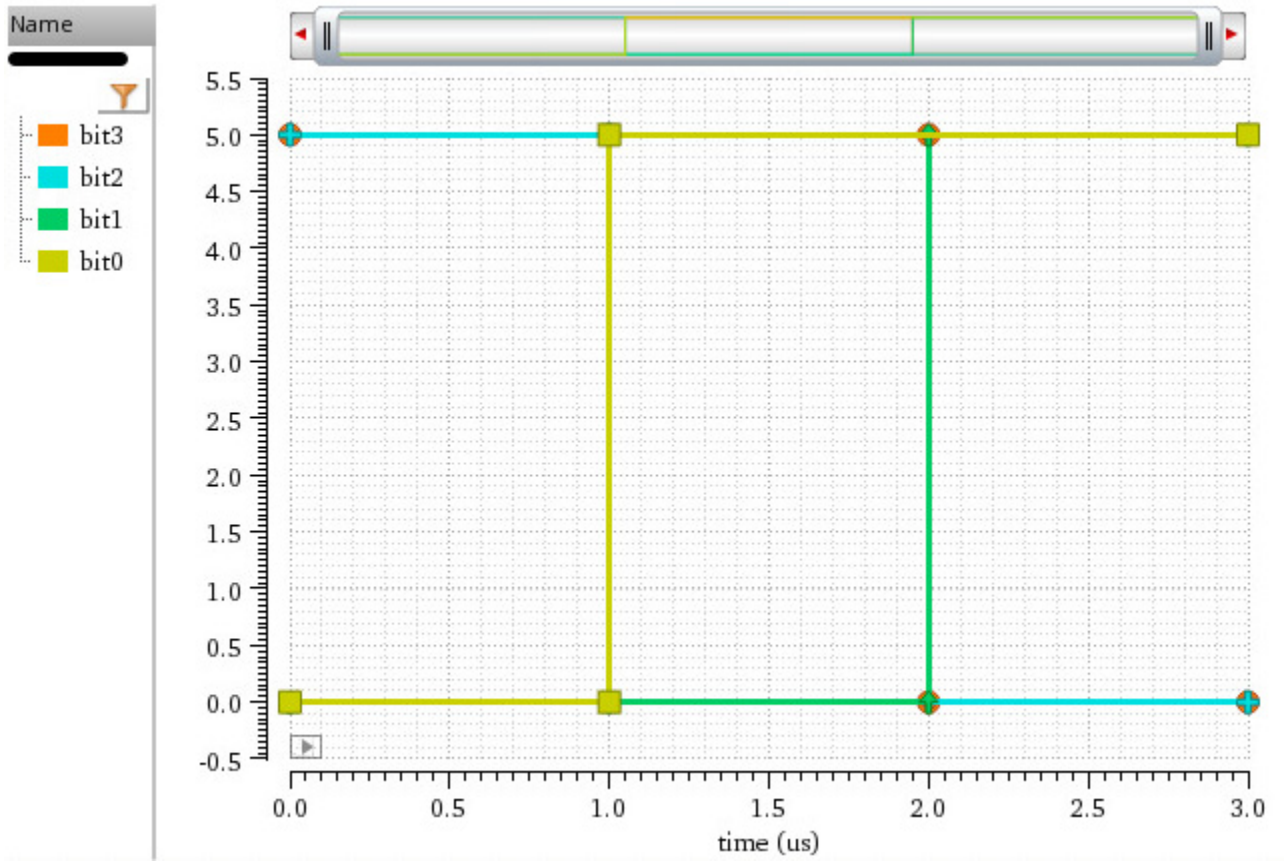
```
    xvec=drCreateVec('double car(list1))
    yvec=drCreateVec('double cadr(list1))
    wave1=drCreateWaveform(xvec yvec)
    drGetWaveformXVec(wave1)~>units = "s"
    drGetWaveformXVec(wave1)~>name = "time"
=>
((0.0 2e-06 2e-06 3e-06)
    (0.0 0.0 5.0 5.0)
)
=> srrVec:0x2d740060
=> srrVec:0x2d740070
=> srrWave:0x2d784030
=> "s"
=> "time"

list2=list(list(0.0 1e-06 1e-06 3e-06) list(5.0 5.0 0.0 0.0))
    xvec=drCreateVec('double car(list2))
    yvec=drCreateVec('double cadr(list2))
    wave2=drCreateWaveform(xvec yvec)
    drGetWaveformXVec(wave2)~>units="s"
    drGetWaveformXVec(wave2)~>name="time"
=>
((0.0 1e-06 1e-06 3e-06)
    (5.0 5.0 0.0 0.0)
)
=> srrVec:0x2d7400a0
=> srrVec:0x2d7400b0
=> srrWave:0x2d784040
=> "s"
=> "time"

list3=list(list(0.0 2e-06 2e-06 3e-06) list(5.0 5.0 0.0 0.0))
    xvec=drCreateVec('double car(list3))
    yvec=drCreateVec('double cadr(list3))
    wave3=drCreateWaveform(xvec yvec)
    drGetWaveformXVec(wave3)~>units="s"
    drGetWaveformXVec(wave3)~>name="time"
=>
((0.0 2e-06 2e-06 3e-06)
    (5.0 5.0 0.0 0.0)
)
```

```
=>srrVec:0x2d7400e0
=> srrVec:0x2d7400f0
=> srrWave:0x2d784050
=> "s"
=> "time"
```

The following example creates a Waveform window and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
awvPlotWaveform(
        win1
        list(wave3 wave2 wave1 wave0)
        ?expr list("bit3" "bit2" "bit1" "bit0")
        ?color list("y6" "y12" "y18" "y24")
        ?lineType list("line" "line" "line" "line")
        ?lineStyle list("solid" "solid" "solid" "solid")
        ?lineThickness list("thick" "thick" "thick" "thick")
        ?showSymbols list(t t t t)
        ?dataSymbol list(9 6 7 3)
        )
```

```
=> t
```



The following examples create digital signals from the specified analog waveforms.

```
dig0=awvAnalog2Digital(wave0 nil nil 2.5 nil "centre")
dig1=awvAnalog2Digital(wave1 nil nil 2.5 nil "centre")
dig2=awvAnalog2Digital(wave2 nil nil 2.5 nil "centre")
dig3=awvAnalog2Digital(wave3 nil nil 2.5 nil "centre")

=> srrWave:0x2d784060
=> srrWave:0x2d784070
=> srrWave:0x2d784080
=> srrWave:0x2d784090
```

The following example creates a digital bus `myBus` from the specified list of digital signals `dig0`, `dig1`, `dig2`, and `dig3`.

```
myBus=awvCreateBus("myBus" list(dig0 dig1 dig2 dig3) "Binary")
=> srrWave:0x2d7840a0
```

The following example creates a Waveform window and returns its window ID.

```
win2=awvCreatePlotWindow()
=> window:4
```

The following example plots the bus `myBus` in the Waveform window `win2`.

```
awvPlotWaveform(
        win2
        list(myBus)
        ?expr list("myBus")
        ?color list("y18")
        )
=> t
```

The following example converts `myBus` into a list of analog signals, each representing a single-bit digital waveform in the bus.

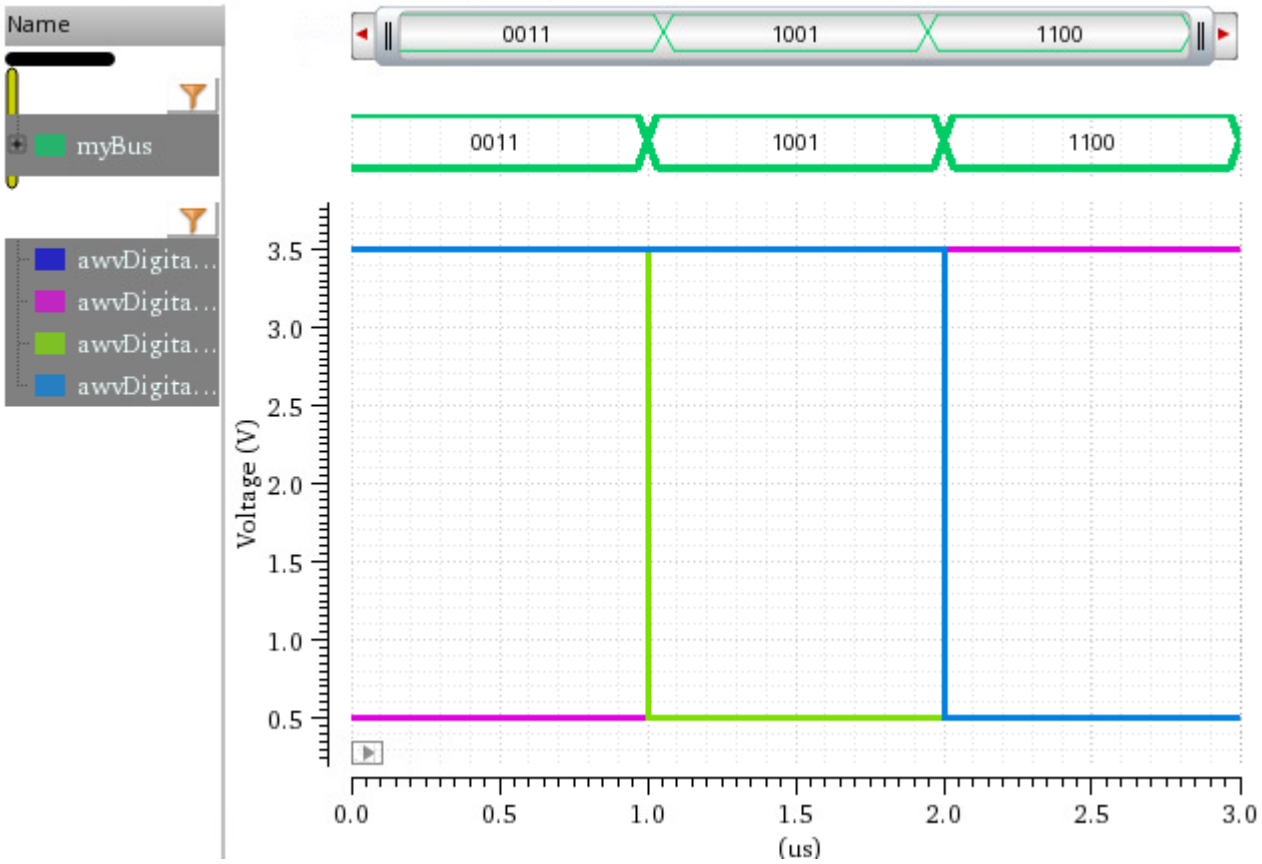
```
analogList=awvDigital2Analog(myBus 3.5 0.5 "vhi+vlo/2.0" ?mode "wavelist"
?outWaveType "pwl")
=>
(srrWave:0x2d7840b0 srrWave:0x2d7840c0 srrWave:0x2d7840d0 srrWave:0x2d7840e0)
```

The following example plots analog signals contained in the list `analogList`. in the current Waveform window.

```
n=0
for(
        i 1 length(analogList)
        plot(nth(n analogList))
        ++n
```

)

awvCreateBus("myBus" list(nil nil nil nil) "Binary")



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* argument. 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.

The function `awvDisableRedraw` will be removed in a future release.

Arguments

w_windowID	Waveform window ID.
g_disable	Specifies whether to disable redraw for the specified Waveform window. Valid values are: ■ `t`: Redraw is disabled for the specified Waveform window. ■ `nil`: Redraw is enabled for the specified Waveform window.

Value Returned

`t`	Redraw status of the specified Waveform window is set successfully.
`nil`	The specified Waveform window does not exist.

Examples

The following example disables redraw status for the current Waveform window.

```
awvDisableRedraw(awvGetCurrentWindow() nil)
```

```
=> t
```

The following example enables redraw status for the current Waveform window.

```
awvDisableRedraw(awvGetCurrentWindow() t)
=> t
```

awvDisplayDate

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

Description

Displays the current data and time in the top-right corner of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.

Value Returned

t	Date and time is displayed.
nil	The specified Waveform window does not exist.

Examples

The following example removes the date and time from the top-right corner of the current Waveform window.

```
win=awvGetCurrentWindow()
=> window:4
awvRemoveDate(win)
=> t
```

The following example removes the date and time from the top-right corner of the specified Waveform window.

```
awvRemoveDate(window(76))
=> t
```

awvDisplayGrid

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

Description

Displays or hides grids in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
g_on	Specifies whether to display grids. Valid values are: ■ `t`: Displays grids. ■ `nil`: Hides grids.
`?subwindow` *x_subwindow*	Identification number of the subwindow (found in the top-right corner of the subwindow) in which grids are to be displayed or hidden. If you do not specify this argument, grids in the current subwindow are displayed or hidden.

Value Returned

`t`	Grids are displayed or hidden in a subwindow of the specified Waveform window.
`nil`	The specified Waveform window does not exist.

Examples

The following example displays grids in the subwindow `1` of the current Waveform window.

```
win = awvGetCurrentWindow()
```

```
=> window:4
awvDisplayGrid(win t ?subwindow 1)
=> t
```

The following example hides grids in the subwindow `4` of the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:4
awvDisplayGrid(win nil ?subwindow 4)
=> t
```

awvDisplaySubwindowTitle

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

Description

Displays the specified title in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
t_title	Specifies the title to be displayed in a subwindow.
?subwindow *x_subwindow*	Identification number of the subwindow (found in the top-right corner of the subwindow) in which the title is to be displayed. If you do not specify this argument, the title is displayed in the current subwindow.

Value Returned

`t`	Title is displayed in a subwindow of the specified Waveform window.
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following example displays the title `Transient Response` in the subwindow `2` of the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:4
awvDisplaySubwindowTitle(win "Transient Response" ?subwindow 2)
=> t
```

The following example displays the title `Noise Analysis` in the current subwindow of the specified Waveform window.

```
awvDisplaySubwindowTitle(window(5) "Noise Analysis")
=> t
```

awvDisplayTitle

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

Description

Displays the specified title in the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
t_title	Specifies the title to be displayed in the Waveform window.

Value Returned

t	Title is displayed in the specified Waveform window.
nil	The specified Waveform window does not exist.

Examples

The following example displays the title `Transient Response` in the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:4
awvDisplayTitle(win "Transient Response")
=> t
```

The following example displays the title `Noise Analysis` in the specified Waveform window.

```
awvDisplayTitle(window(5) "Noise Analysis")
=>t
```

awvEraseWindowMenuCB

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

Description

Deletes all the objects from a Waveform window. The function is defined in `dfII/etc/context/awv.cxt`.

This function works only on the Waveform windows and objects that are created using SKILL calls.

Arguments

None

Value Returned

`t`	Objects are deleted successfully from the Waveform window.

Examples

The following example opens the simulation results stored in the specified directory.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

The following example plots the waveforms of signals `net10` and `out` in a Waveform window.

```
plot(v("net10" ?result "tran-tran") v("out" ?result "tran-tran"))
=> t
```

The following function deletes waveform objects from the Waveform window.

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

awvEval

```
awvEval(
     t_placeholder
     t_expr
     )
     => t_expr
```

Description

Returns the second argument. The first argument is only a placeholder argument.

Arguments

t_placeholder	Placeholder argument.
t_expr	Expression to be evaluated. The function returns this expression.

Value Returned

t_expr	Second argument of the function.

Examples

The following example creates a Waveform window.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens the simulation results stored in the specified directory.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

The following example returns ID of the resulting waveform evaluated as per the expression. Signals `out` and `net10` are stored in the `tran-tran` results of the results directory `/servers/user/design/ampsim.raw`.

```
waveform=v("out" ?result "tran-tran")-v("net10" ?result "tran-tran")
=> srrWave:0x36167040
```

This waveform object ID is passed as the second argument of the function `awvEval`. The function returns the same waveform object ID.

```
awvEval("expr" waveform)
=> srrWave:0x36167040
```

awvExitWindowFunctionAdd

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

Description

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

The list of functions is empty by default.

Arguments

u_function	Function to be added to the list of exit functions. Callback parameter list: *w_windowID*

Value Returned

t	Function is added to the list.
nil	Function cannot be added to the list because of an error.

Examples

None

awvExitWindowFunctionDel

```
awvExitWindowFunctionDel(
     u_function
     )
     => t / nil
```

Description

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

Arguments

u_function	Function to be deleted from the list of exit functions.

Value Returned

t	Function is deleted from the list.
nil	Function cannot be deleted from the list because of an error.

Examples

The following example deletes the exit function myExitFunc from the list of exit functions.

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

awvExitWindowFunctionGet

```
awvExitWindowFunctionGet(
     )
     => l_exitFunctionList / nil
```

Description

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

This list is empty by default.

Arguments

None

Value Returned

l_exitFunctionList	A list containing names of exit functions.
nil	There are no exit functions.

Examples

The following example returns a list of names of current exit functions.

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

awvEyeCross

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

Description

Returns a waveform showing x-axis values where an eye diagram crosses the specified threshold value on y axis.

Arguments

w_waveform	The eye diagram waveform.
n_start	Start time.
n_stop	Stop time.
n_eyePeriod	Eye period of the eye diagram.
n_threshold	Threshold value on y axis.
x_edgeType	Type of crossing. Valid values are: `rising`, `falling`, and `either`. If you do not specify this argument, default value `either` is used.
x_ignoreStart	The time before which any crosses occurred in the eye diagram are ignored.
x_ignoreEnd	The time after which any crosses occurred in the eye diagram are ignored. This is an optional argument.

Value Returned

o_waveform	Returns a waveform showing cross numbers on x axis and cross time on y axis.
nil	The specified eye diagram does not exist or the output waveform object cannot be created because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
win=awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified results directory.

```
openResults("/servers/user/design/prbs.raw")
=> "/servers/user/design/prbs.raw"
```

The following example lists the available results in the currently open results directory.

```
results()
=> tran()
```

The following example selects the `tran` result of the results directory.

```
selectResult('tran)
=> stdobj@0x32e1bf20
```

The following example creates a waveform object, representing the waveform of the signal `jitter`, which is stored in the `tran` result of the results directory.

```
jitter=v("jitter")
=> srrWave:0x36b8f020
```

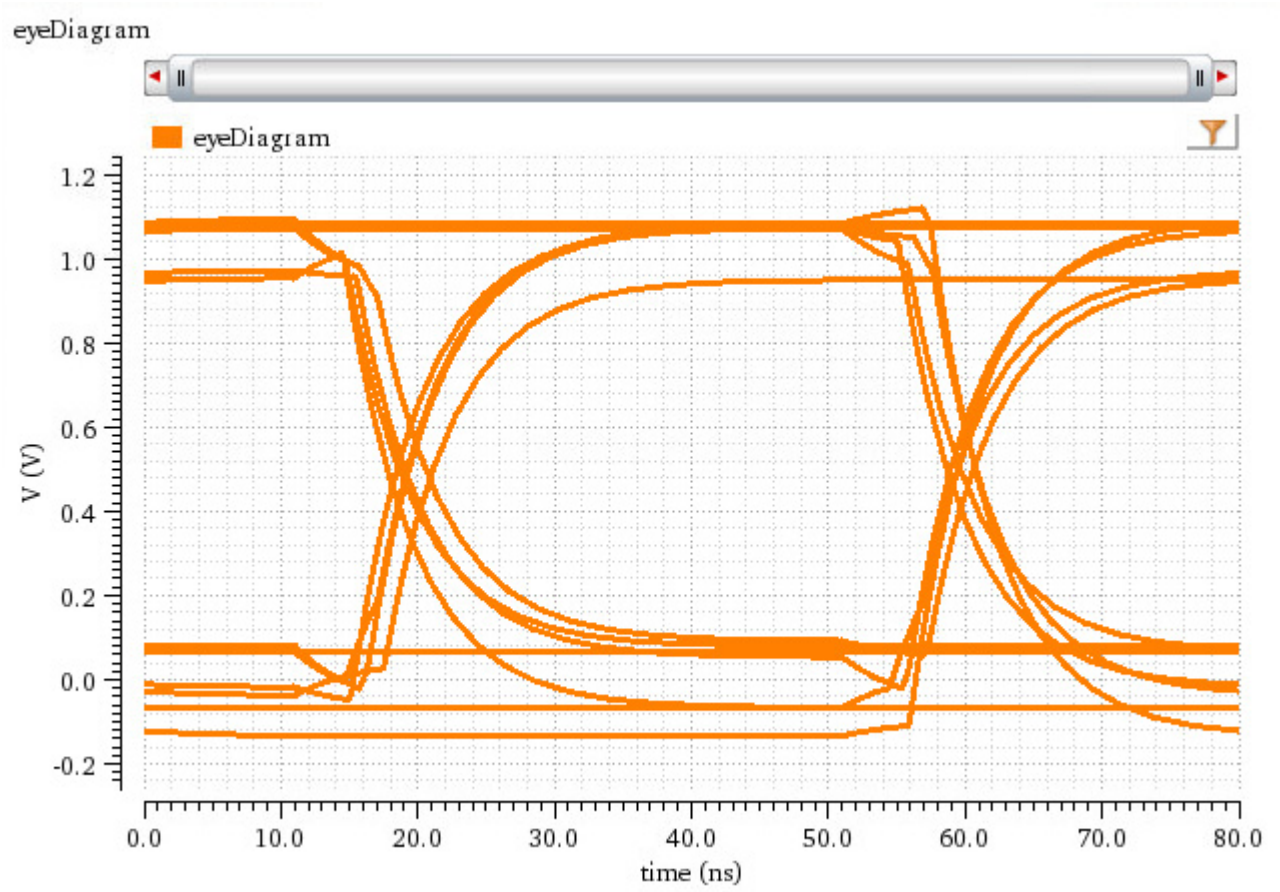
The following example creates a waveform object `eye`, representing the eye diagram that is created by applying `eyeDiagram` function on the `jitter` signal.

```
eye=eyeDiagram(jitter 200n 1.4u 2*40n ?autoCenter t)
=> srrWave:0x36b8f030
```

The following example plots the eye diagram in the specified Waveform window.

```
awvPlotWaveform(
                win
                list(eye)
                ?expr list("eyeDiagram")
                ?color list("y6")
```

```
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                )
=> t
```



The following example creates another Waveform window and returns its window ID.

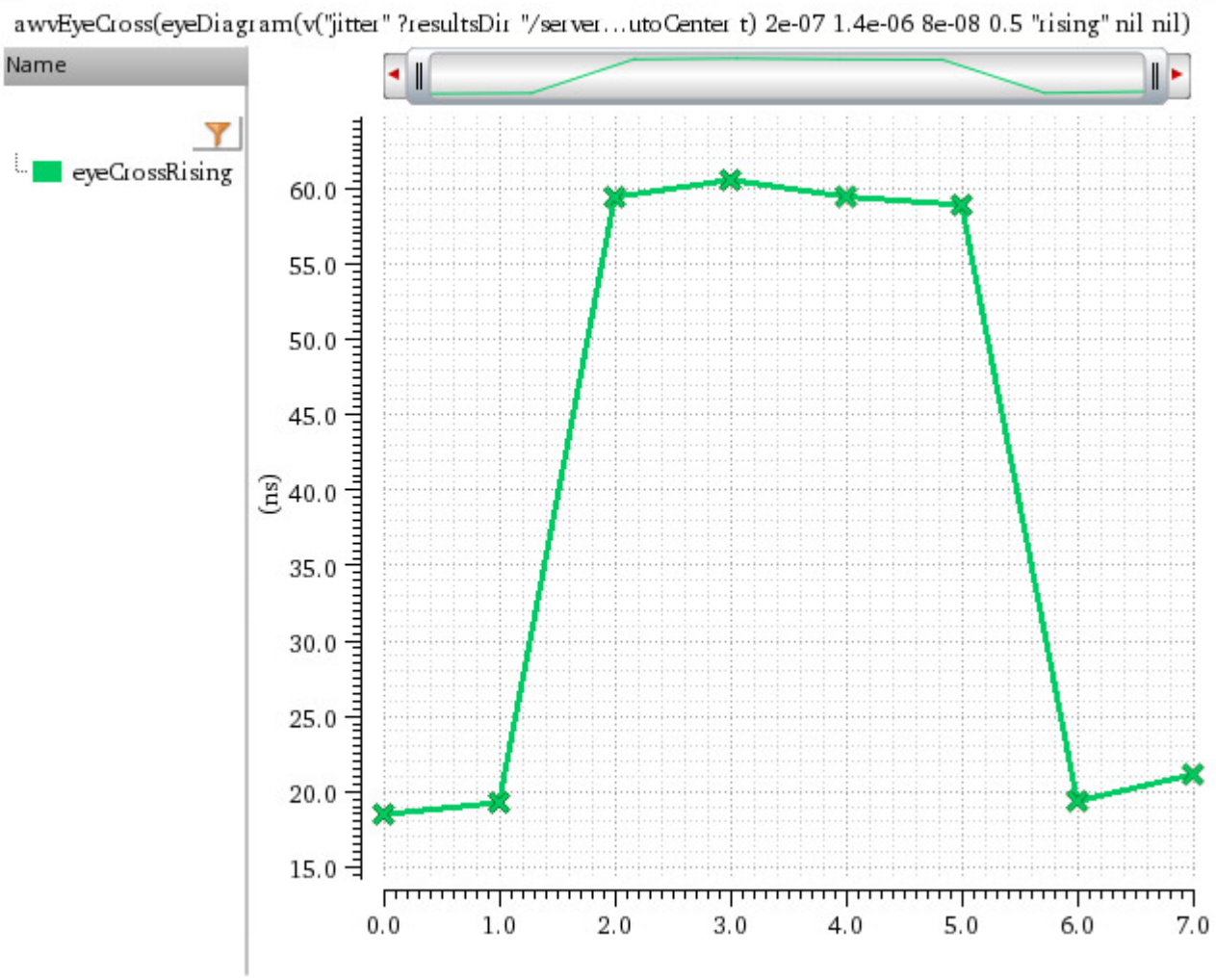
```
win=awvCreatePlotWindow()
=> window:4
```

The following example creates a waveform object `eyeCrossRising`, which shows time when the rising edge of the eye diagram `eye` crosses the threshold value `0.5` plotted on y axis against the cross values plotted on x axis

```
eyeCrossRising=awvEyeCross(eye 200n 1.4u 80n 0.5 "rising")
=> srrWave:0x36b4c050
```

The following example plots the waveform object `eyeCrossRising`.

```
awvPlotWaveform(
                win list(eyeCrossRising)
                ?expr list("eyeCrossRising")
                ?color list("y18")
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("x")
                )
=> t
```



The following example creates a Waveform window and returns its window ID.

```
win=awvCreatePlotWindow()
=> window:5
```

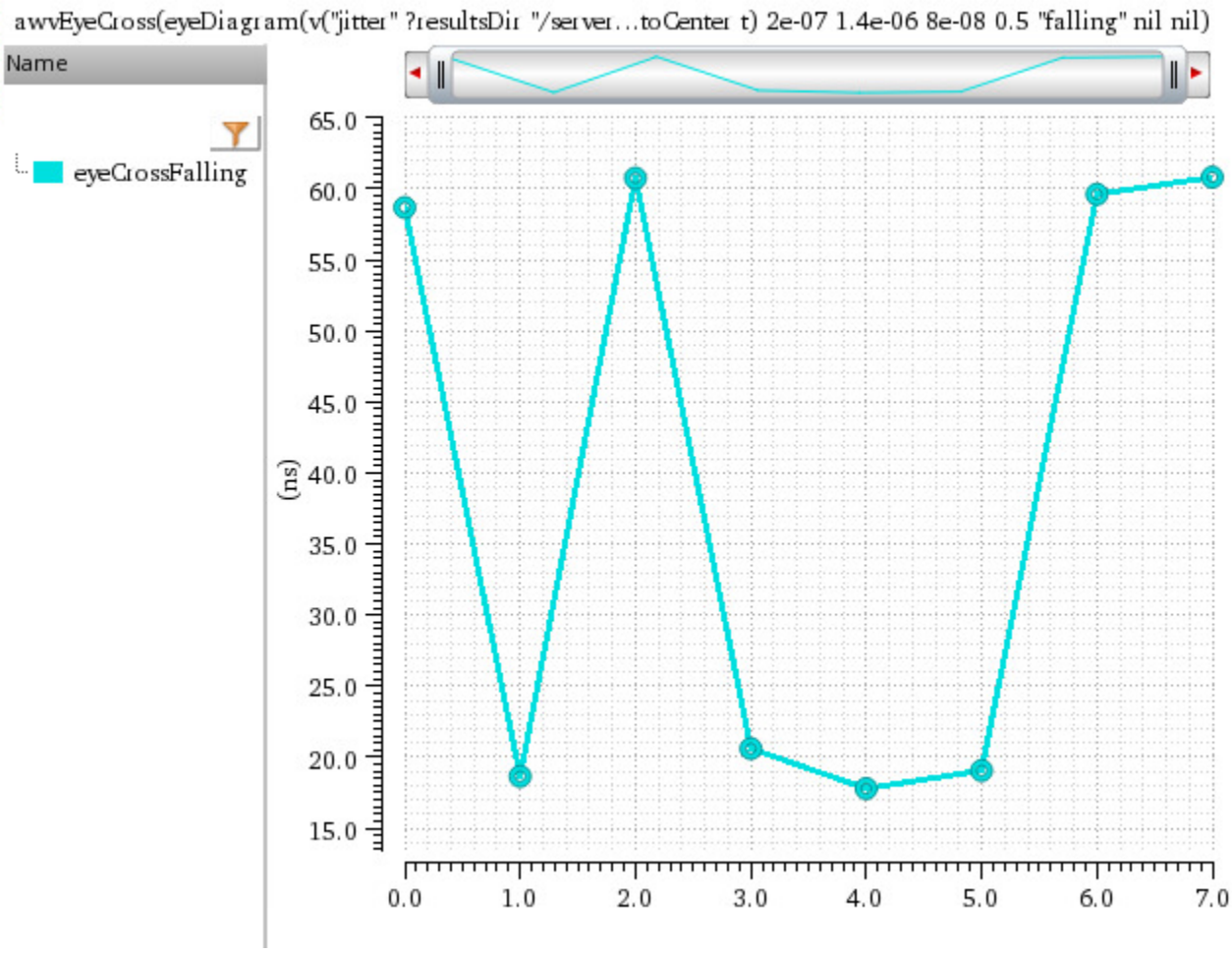
The following example creates a waveform object `eyeCrossFalling`, which shows time when the falling edge of the eye diagram `eye` crosses the threshold value `0.5` plotted on y axis against the cross values plotted on x axis.

```
eyeCrossFalling=awvEyeCross(eye 200n 1.4u 80n 0.5 "falling")
=> srrWave:0x36b4c070
```

The following example plots the waveform object `eyeCrossFalling`.

```
awvPlotWaveform(
                win list(eyeCrossFalling)
                ?expr list("eyeCrossFalling")
                ?color list("y12")
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("O")
                )
```

```
=> t
```



The following example creates a Waveform window and returns its window ID.

```
win=awvCreatePlotWindow()
=> window:6
```

The following example creates a waveform object `eyeCrossEither1`, which shows time between `0` and `40ns` when any edge (either rising or falling) of the eye diagram `eye` crosses the threshold value `0.5` plotted on y axis against the cross values plotted on x axis.

```
eyeCrossEither1=awvEyeCross(eye 200n 1.4u 80n 0.5 "either" 0 40n)
=> srrWave:0x36b4c090
```

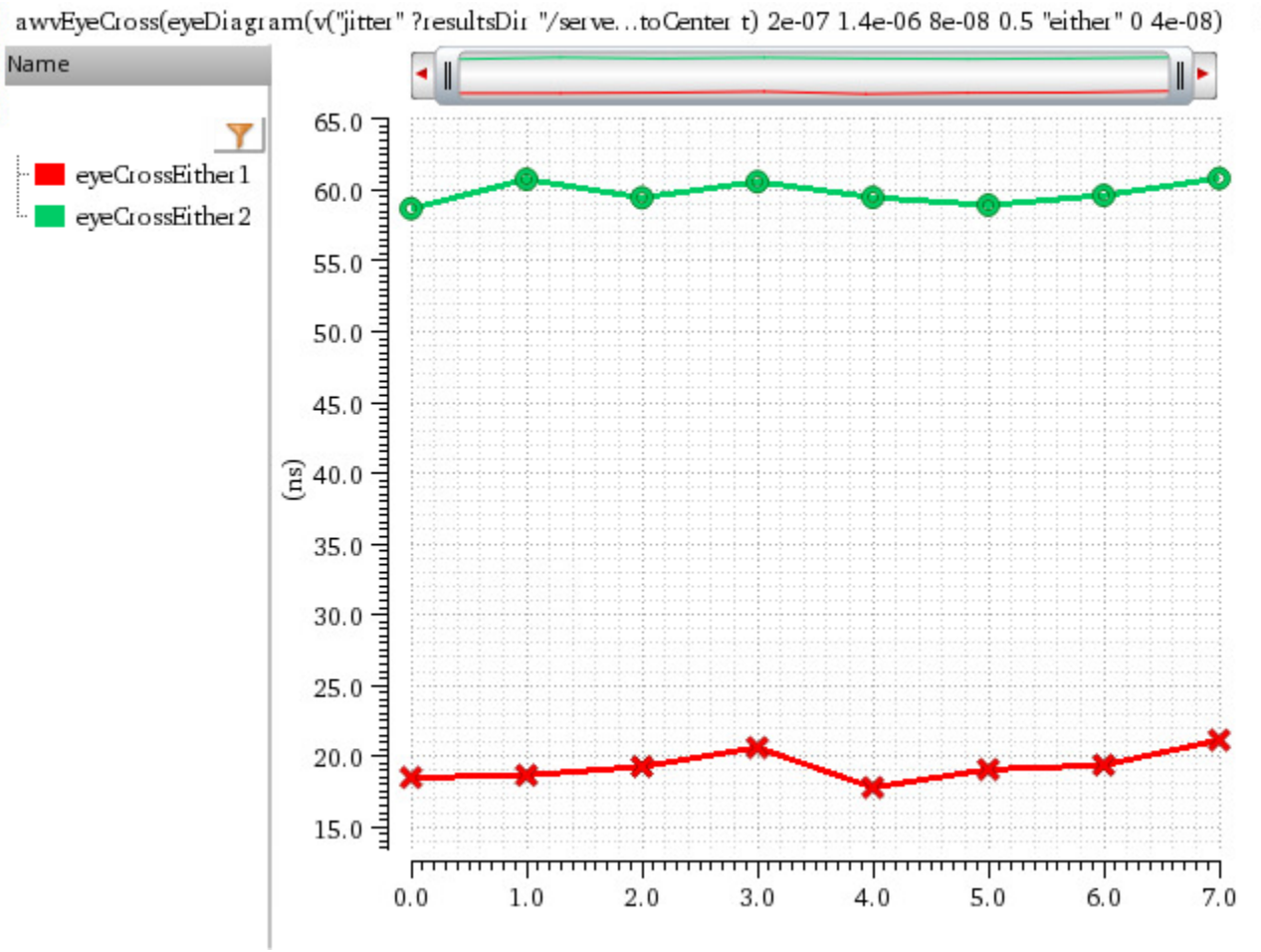
The following example creates a waveform object `eyeCrossEither2`, which shows time between `40ns` and `80ns` when any edge (either rising or falling) of the eye diagram `eye` crosses the threshold value `0.5` plotted on y axis against the cross values plotted on x axis.

```
eyeCrossEither2=awvEyeCross(eye 200n 1.4u 80n 0.5 "either" 40n 80n)
=> srrWave:0x36b4c0b0
```

The following example plots the waveform objects, `eyeCrossEither1` and `eyeCrossEither2`.

```
awvPlotWaveform(
                win
                list(eyeCrossEither1 eyeCrossEither2)
                ?expr list("eyeCrossEither1" "eyeCrossEither2")
                ?color list("y1" "y66")
                ?lineType list("line" "line")
                ?lineStyle list("solid" "solid")
                ?lineThickness list("thick" "thick")
                ?showSymbols list(t t)
                ?dataSymbol list("X" "O")
                )
```

```
=> t
```



awvGetAssertName

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

Description

Returns the name of the assert defined in violation data.

Arguments

o_waveform	Name of the waveform object.

Value Returned

t_assertName	Name of the assert.
nil	Name of the assert cannot be returned because of an error.

Examples

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
Interactive.16/psf")
=> "/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
Interactive.16/psf"
```

The following example lists results available in the selected results directory.

```
results()
=> tran("tranViolations-violations" tranOp model instance output
    designParamVals primitives subckts variables
)
```

The following example selects the result `tranViolations-violations`.

```
selectResults("tranViolations-violations")
=> stdobj@0x30fbade8
```

The following example lists the outputs available in the selected result.

```
outputs()
=> ("OOB_FWD_ncap_25_lp.C0.violation1")
```

The following example gets data.

```
wave=getData("OOB_FWD_ncap_25_lp.C0.violation1")
=> dr:0x30eea300
```

The following example returns the name of the assert defined in the violation data.

```
awvGetAssertName(wave)
=> ("OOB_FWD_ncap_25_lp")
```

awvGetCurrentSubwindow

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

Description

Returns the identification number of the current subwindow in the specified Waveform window.

Arguments

w_windowID	Waveform window ID.

Value Returned

x_subwindow	The identification number of the current subwindow (found in the top-right corner of the subwindow) of the specified Waveform window.
nil	The specified Waveform window does not exist.

Examples

The following example removes the title from the subwindow 2 of the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:4
awvRemoveSubwindowTitle(win ?subwindow 2)
=> t
```

The following example removes the title from the current subwindow of the specified Waveform window.

```
awvRemoveSubwindowTitle(window(5))
```

awvGetCurrentWindow

```
awvGetCurrentWindow(
     )
     => w_windowID / nil
```

Description

Returns the window ID of the current Waveform window.

Arguments

None

Value Returned

w_windowID	Window ID of the current Waveform window.
nil	No Waveform window is currently open.

Examples

The following example returns the window ID of the current Waveform window.

```
awvGetCurrentWindow()
=> window:8
```

awvGetDisplayMode

```
awvGetDisplayMode(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => t_mode / nil
```

Description

Returns display mode of a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow (found in the top-right corner of the subwindow) whose display mode is to be returned. If you do not specify this argument, display mode of the current subwindow is returned.

Value Returned

t_mode	Display mode of the subwindow. Valid values are: ■ `composite` ■ `smith` ■ `strip`
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following example returns display mode of the subwindow `2` of the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:8
```

```
awvGetDisplayMode(win ?subwindow 2)
=> "smith"
```

The following example returns display mode of the current subwindow of the specified Waveform window.

```
awvGetDisplayMode(window(9))
=> "composite"
```

awvGetDrawStatus

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

Description

Returns the draw status of the specified Waveform window.

Arguments

w_windowID	ID of the waveform window whose draw status is to be returned.

Value Returned

t	
nil	The specified Waveform window does not exist.

Examples

The following example returns the draw status of the current Waveform window.

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

The following example returns the draw status of the specified Waveform window.

```
awvGetDrawStatus(window(3))
=> t
```

awvGetHiWindow

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

Description

Returns the ID of the waveform HI window corresponding to the specified Waveform window ID.

This function is primarily used to customize menu using HI calls.

Arguments

w_windowID	Waveform window ID.

Value Returned

w_HIWindowID	Returns the HI window ID.
nil	Indicates an error.

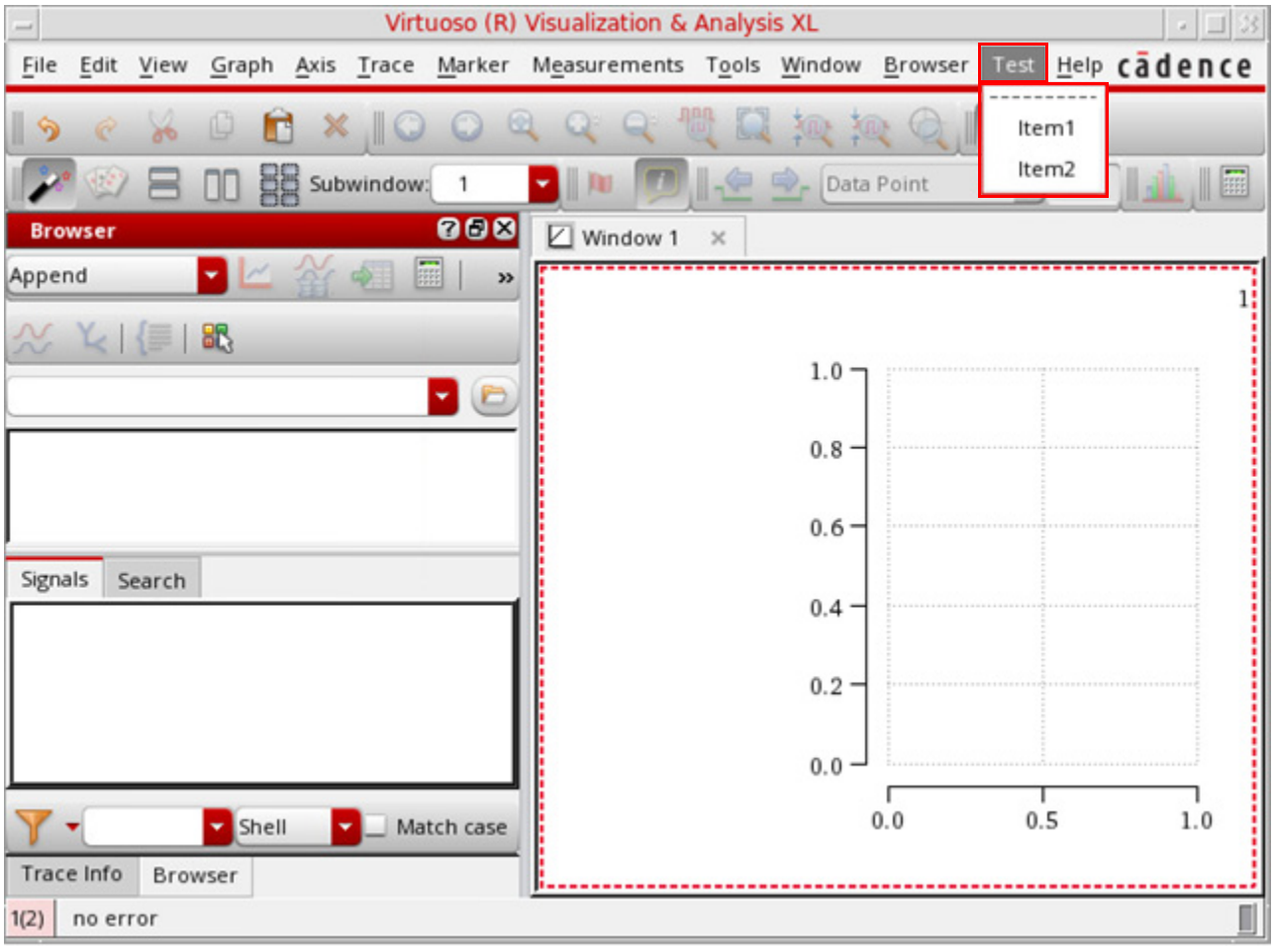
Examples

The following example adds a procedure `myCreateMenu`, which creates a new banner menu *Test* in the Results Browser. The *Test* menu has two options, *Item1* and *Item2*.

- When you click *Item1*, the message "`Item 1 Selected`" is printed in the CIW.
- When you click *Item2*, the message "`Item 2 Selected`" is printed in the CIW.

```
myCreateMenu
=> t

windowId=awvGetHiWindow(awvGetCurrentWindow())
procedure(
    myCreateMenu(windowId)
    win=awvGetHiWindow(windowId)
    let((item1 item2)
        item1=hiCreateMenuItem(
```

```
            ?name 'item1
            ?itemText "Item1"
            ?callback "println(\"Item 1 Selected\")"
        )
        item2=hiCreateMenuItem(
            ?name 'item2
            ?itemText "Item2"
            ?callback "println(\"Item 2 Selected\")"
        )
        hiCreatePulldownMenu( 'myMenu "Test" list(item1 item2))
        hiInsertBannerMenu(win myMenu hiGetNumMenus(win))
        )
    )
    awvInitWindowFunctionAdd('myCreateMenu)
=> myCreateMenu
=> t
```



The following message is printed in CIW when you click *Item1* from the *Test* menu.

```
println("Item 1 Selected")
"Item 1 Selected"
```

The following message is printed in CIW when you click *Item2* from the *Test* menu.

```
println("Item 2 Selected")
"Item 2 Selected"
```

awvGetInitializationTimeout

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

Description

Returns the timeout period in second set for ADE to establish connection with Virtuoso Visualization and Analysis XL. The default value of timeout period is `120` second, which means that ADE keeps trying to establish a connection with Virtuoso Visualization and Analysis XL for up to `120` second.

Arguments

None

Value Returned

x_timeOut	Timeout period in second.

Examples

The following example returns the timeout period set for ADE to establish connection with Virtuoso Visualization and Analysis XL.

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

awvGetLegendPos

```
awvGetLegendPos(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => t_legendPosition / nil
```

Description

Returns the position of the trace legend in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, position of the trace legend in the current subwindow is returned.

Value Returned

t_legendPosition	Position of the trace legend in a subwindow. Valid values are: ■ `above` ■ `inside` ■ `left`
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following examples return the position of the trace legend in subwindows `1`, `3`, and `5` in the specified Waveform window, respectively.

```
awvGetLegendPos(window(4) ?subwindow 1)
=> "left"
```

```
awvGetLegendPos(window(4) ?subwindow 3)
=> "inside"
awvGetLegendPos(window(4) ?subwindow 5)
=> "above"
```

The following example returns the position of the trace legend in the current subwindow of the specified Waveform window.

```
awvGetLegendPos(window(4))
=> "above"
```

awvGetOnSubwindowList

```
awvGetOnSubwindowList(
     w_windowID
     [ ?all g_all ]
     )
     => l_onSubwindows / nil
```

Description

Returns a list of subwindows that are being used in the specified Waveform window.

This list includes only those subwindows whose display and update statuses are turned on. To get a list of all subwindows whose display statuses are on, regardless of their update statuses, set `?all` to `t`.

Arguments

w_windowID	Waveform window ID.
?all *g_all*	Specifies whether the function returns a list of all subwindows whose display statuses are on, regardless of their update statuses. Valid values are: ■ `t`: The functions returns a list of all subwindows whose display statuses are on, regardless of their update statuses. ■ `nil`: The function returns a list of only those subwindows whose update and display statuses are turned on. This is the default value.

Value Returned

l_subwindows	A list of subwindows whose display statuses are on.
`nil`	The specified Waveform window does not exist.

Examples

The following example returns a list of subwindows whose update and display statuses are turned on in the Waveform window `4`.

```
awvGetOnSubwindowList(window(4))
=> (1 2 3 4)
```

The following example turns off the update status of subwindow `2` in the Waveform window `4`.

```
awvSetUpdateStatus(window(4) nil ?subwindow 2)
=> t
```

The following example turns off the update status of subwindow `4` in the Waveform window `4`.

```
awvSetUpdateStatus(window(4) nil ?subwindow 4)
=> t
```

The following example turns off the display status of subwindow `2` in the Waveform window `4`.

```
awvSetDisplayStatus(window(4) nil ?subwindow 2)
=> t
```

The following example returns only subwindows `1` and `3` because only they have both their update and display statuses on.

```
awvGetOnSubwindowList(window(4))
=> (1 3)
```

The following example returns subwindows `1`, `3`, and `4` because they have their display statuses on, regardless of their update statuses.

```
awvGetOnSubwindowList(window(43) ?all t)
=> (1 3 4)
```

awvGetPlotStyle

```
awvGetPlotStyle(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => t_plotStyle / nil
```

Description

Returns the plotting style for waveforms in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow (found in the top-right corner of the subwindow) whose plotting style is to be returned. If you do not specify this argument, plotting style of the current subwindow is returned.

Value Returned

t_plotStyle	Plotting style of the subwindow.
nil	The specified Waveform window or the subwindow does not exist.

Examples

The following example returns the plotting style set to subwindow 2 in the current Waveform window.

```
awvGetPlotStyle(awvGetCurrentWindow() ?subwindow 2)
=> "points"
```

The following example returns the plotting style set to subwindow 4 in the specified Waveform window.

```
awvGetPlotStyle(window(3) ?subwindow 4)
=> "line"
```

awvGetScalarFromWave

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

Description

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

Arguments

o_waveform	The waveform whose scalar value is to be returned.

Value Returned

n_yValue	Y-axis value of the point in a single-point waveform.
o_waveform	Waveform ID of the input waveform if it has multiple points.

Examples

The following example creates a single-point waveform `wave`.

```
xVec=drCreateVec('double 10)
=> srrVec:0x360c77a0
yVec=drCreateVec('double 10)
=> srrVec:0x360c77b0
drSetElem(xVec 0 10)
=> t
drSetElem(yVec 0 10)
=> t
wave=drCreateWaveform(xVec yVec)
=> srrWave:0x3603d200
```

The following example returns the scalar value of the single-point waveform `wave`.

```
awvGetScalarFromWave(wave)
=> 10.0
```

The following example creates a waveform that has multiple points.

```
xVec=drCreateVec('double 10)
yVec=drCreateVec('double 10)
for(i 1 10
    drSetElem(xVec i i)
    drSetElem(yVec i i*i-19*1+100)
)
=> srrVec:0x360c7af0
=> srrVec:0x360c7b00
=> t
wave=drCreateWaveform(xVec yVec)
=> srrWave:0x3603d2b0
```

When you run the function `awvGetScalarFromWave` on the waveform `wave`, the function returns ID of the waveform.

```
awvGetScalarFromWave(wave)
=> srrWave:0x3603d2b0
```

awvGetSelectedTraceWaveforms

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

Description

Returns a list of waveform IDs of the traces that are selected in a subwindow of the specified Waveform window.

This function does not support trace groups.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

l_waveformList	A list of waveform IDs of the traces selected in a subwindow.
nil	Either no traces are currently selected or the specified Waveform window or subwindow does not exist.

Examples

The following examples returns a list of waveforms IDs of the traces that are currently selected in the subwindow 3 of the specified Waveform window.

```
awvGetSelectedTraceWaveforms(window(10) ?subwindow 3)
=> (srrWave:0x397fa040 srrWave:0x397fa060 srrWave:0x397fa070)
```

awvGetSmithModeType

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

Description

Returns the type of Smith display of a subwindow in the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, type of Smith display of the current subwindow is returned.

Value Returned

t_type	Type of Smith display of the subwindow. Valid values are: ■ `polar` ■ `impedance` ■ `admittance` ■ `immittance`
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following examples return the type of Smith display of subwindows `1`, `3`, `5`, and `7` in the specified Waveform window, respectively.

```
awvGetSmithModeType(window(90) ?subwindow 1)
```

```
=> "polar"
awvGetSmithModeType(window(90) ?subwindow 3)
=> "impedance"
awvGetSmithModeType(window(90) ?subwindow 5)
=> "admittance"
awvGetSmithModeType(window(90) ?subwindow 7)
=> "immittance"
```

The following example returns the type of Smith display of the current subwindow in the specified Waveform window.

```
awvGetSmithModeType(window(89))
=> "impedance"
```

awvGetStripNumberOfSelectedTrace

```
awvGetStripNumberOfSelectedTrace(
     w_windowID
     )
     => n_stripNumber / nil / -1
```

Description

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

Arguments

w_windowID	Waveform window ID.

Value Returned

n_stripNumber	Strip number of the selected trace.
nil	The specified Waveform window does not exist.
-1	No trace is plotted in the specified Waveform window.

Examples

The following example returns the strip number of the trace selected in the current Waveform window.

```
win= awvGetCurrentWindow()
=> window:3
awvGetStripNumberOfSelectedTrace(win)
=> 9
```

The following example returns the strip number of the currently selected trace in the specified Waveform window.

```
awvGetStripNumberOfSelectedTrace(window(4))
=> 7
```

awvGetStripNumbersList

```
awvGetStripNumbersList(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => l_stripNumbersList / nil
```

Description

Returns the strip numbers of traces plotted in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

l_stripNumbersList	A list containing strip numbers of the traces plotted in a subwindow of the specified Waveform window.
nil	The specified Waveform window or subwindow does not exist.

Examples

The following example returns strip numbers of the traces plotted in the subwindow 4 of the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:4
awvGetStripNumbersList(win ?subwindow 4)
=>
(1 3 4 5 6
```

```
    7 8 9 10 11
    12 13 14 2
)
```

The following example returns the strip numbers of the traces plotted in the subwindow `3` of the specified Waveform window.

```
awvGetStripNumbersList(window(5) ?subwindow 3)
=> (14 1 2 3)
```

awvGetSubwindowList

```
awvGetSubwindowList(
     w_windowID
     [ ?all g_all ]
     )
     => l_subwindows / nil
```

Description

Returns a list of subwindows whose update statuses are turned on.

To return a list of all subwindows, including those whose updates statuses are turned off, set the `?all` argument to `t`.

Arguments

w_windowID	Waveform window ID.
`?all` *g_all*	Specifies whether the function returns a list of all subwindows of the specified Waveform window. Valid values are: ■ `t`: The functions returns a list of all subwindows of the Waveform window. ■ `nil`: The function returns a list of only those subwindows whose update statuses are turned on. This is the default value.

Value Returned

l_subwindows	A list of subwindows in the specified Waveform window.
`nil`	The specified Waveform window does not exist.

Examples

The following example returns a list of all subwindows of the current Waveform window. The list also includes the subwindows whose update statuses are turned off.

```
win = awvGetCurrentWindow()
=> window:5
```

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

The following example returns a list of only those subwindows whose update statuses are turned on.

```
win = awvGetCurrentWindow()
=> window:6
awvGetSubwindowList(win ?all nil)
=> (2 3)
```

awvGetSubwindowStripCount

```
awvGetSubwindowStripCount(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => n_stripCount / nil
```

Description

Returns the total number of strips plotted in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

n_stripCount	Number of strips plotted in a subwindow of the specified Waveform window.
nil	The specified Waveform window or subwindow does not exist.

Examples

The following example returns the total number of strips plotted in the subwindow 3 of the current Waveform window.

```
win=awvGetCurrentWindow()
=> window:5
awvGetSubwindowStripCount(win ?subwindow 3)
=> 14
```

The following example returns the total numbers of strips plotted in the subwindow 2 of the specified Waveform window.

```
awvGetSubwindowStripCount(window(6) ?subwindow 2)
```

```
=> 9
```

awvGetSubwindowTitle

```
awvGetSubwindowTitle(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => t_subwindowTitle / nil
```

Description

Returns the title of a subwindow in the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow (found in the top-right corner of the subwindow) whose title is to be returned. If you do not specify this argument, the title of the current subwindow is returned.

Value Returned

t_subwindowTitle	Title of a subwindow in the specified Waveform window.
nil	The specified Waveform window or subwindow does not exist.

Examples

The following example returns the title of the current subwindow of the current Waveform window.

```
awvGetSubwindowTitle(awvGetCurrentWindow() ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> "Transient Analysis 'tran': time = (0 s -> 100 ns)"
```

The following example sets the title of the current subwindow to `Transient Response`.

```
awvDisplaySubwindowTitle(awvGetCurrentWindow() "Transient Response" ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
```

The following function now returns the title of the current subwindow that is set using the `awvDisplaySubwindowTitle` function.

```
awvGetSubwindowTitle(awvGetCurrentWindow() ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> "Transient Response"
```

awvGetUnusedEntityList

```
awvGetUnusedEntityList(
     w_windowID
     [ ?subwindow x_subwindow ]
     [ ?total x_total ]
     )
     => l_waveformIndexes / nil
```

Description

Returns a list of integers that have not already been used to identify waveforms in a subwindow.

You can specify the total number of unused waveform indexes to include in the return value

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	
	Identification number of the subwindow (found in the top-right corner of the subwindow) in which unused waveform indexes are to be returned. If you do not specify this argument, the unused waveform indexes in the current subwindow are returned.
?total *x_total*	Specifies the total number of unused waveform indexes to include in the return value *l_waveformIndexes* of the function.

Value Returned

l_waveformIndexes	A list of next lowest integers that have not been used to identify existing waveforms in a subwindow of the specified Waveform window.
nil	Indicates an error.

Examples

The following example returns a list of `7` unused waveform indexes that have not been used in subwindow `2` of the current Waveform window.

```
awvGetUnusedEntityList(awvGetCurrentWindow() ?subwindow 2 ?total 7)
=>
(27 28 29 30 31
    32 33
)
```

awvGetWaveNameList

```
awvGetWaveNameList(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => l_infoList / nil
```

Description

Returns a list containing numbers and names of the waveforms plotted in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow (found in the top-right corner of the subwindow). If you do not specify this argument, the current subwindow is used.

Value Returned

l_infoList	A list containing numbers and names of the waveforms.
nil	The specified Waveform window or subwindow does not exist.

Examples

The following example returns a list of numbers and names of the waveforms plotted in the specified subwindow of the current Waveform window.

```
win=awvGetCurrentWindow()
=> window:8
awvGetWaveNameList(win ?subwindow 3)
=>
((39 33 34 35 36
    37 38 27 28 29
    30 31 32
```

```
    )
    ("/OUTP (modelFiles=gpdk090.scs:NN,vdd=2.00e+00,temperature=2.70e+01)" "/OUTP
(modelFiles=gpdk090.scs:FF,vdd=1.80e+00,temperature=-4.00e+01)" "/OUTP
(modelFiles=gpdk090.scs:FF,vdd=1.80e+00,temperature=0.00e+00)" "/OUTP
(modelFiles=gpdk090.scs:FF,vdd=1.80e+00,temperature=1.25e+02)" "/OUTP
(modelFiles=gpdk090.scs:FF,vdd=2.20e+00,temperature=-4.00e+01)"
    "/OUTP (modelFiles=gpdk090.scs:FF,vdd=2.20e+00,temperature=0.00e+00)" "/OUTP
(modelFiles=gpdk090.scs:FF,vdd=2.20e+00,temperature=1.25e+02)" "/OUTP
(modelFiles=gpdk090.scs:SS,vdd=1.80e+00,temperature=-4.00e+01)" "/OUTP
(modelFiles=gpdk090.scs:SS,vdd=1.80e+00,temperature=0.00e+00)" "/OUTP
(modelFiles=gpdk090.scs:SS,vdd=1.80e+00,temperature=1.25e+02)"
    "/OUTP (modelFiles=gpdk090.scs:SS,vdd=2.20e+00,temperature=-4.00e+01)" "/OUTP
(modelFiles=gpdk090.scs:SS,vdd=2.20e+00,temperature=0.00e+00)" "/OUTP
(modelFiles=gpdk090.scs:SS,vdd=2.20e+00,temperature=1.25e+02)"
    )
)
```

The following example returns a list of numbers and names of the waveforms plotted in the subwindow 2 of the current Waveform window.

```
win=awvGetCurrentWindow()
=> window:9
awvGetWaveNameList(win ?subwindow 2)
=>
((1 2 5)
    ("in_p" "out" "R2:pwr")
)
```

The following example returns a list of the numbers and names of the waveforms plotted in the current subwindow of the specified Waveform window.

```
awvGetWaveNameList(window(9))
=>
((10 11 9)
    ("V0:p" "net10" ":pwr")
)
```

awvGetWindowList

```
awvGetWindowList(
     )
     => l_windows / nil
```

Description

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

Arguments

None

Value Returned

l_windows	A list containing IDs of Waveform windows.
nil	No Waveform windows are currently open.

Examples

The following example returns a list of IDs of all Waveform windows associated with the current process.

```
awvGetWindowList()
=> (window:3 window:4 window:5)
```

awvGetWindowTitle

```
awvGetWindowTitle(
     w_windowID
     )
     => t_windowTitle / nil
```

Description

Returns the title of a subwindow in the specified Waveform window.

Arguments

w_windowID	ID of the Waveform window whose title is to be returned.

Value Returned

t_windowTitle	Title of the specified Waveform window.
nil	The specified Waveform window does not exist.

Examples

The following example returns the title of the current Waveform window.

```
awvGetWindowTitle(awvGetCurrentWindow())
=> "Window 96"
```

The following example sets the title of the current Waveform window to `Transient Analysis`.

```
awvDisplayTitle(awvGetCurrentWindow() "Transient Analysis")
=> t
```

The following function now returns the title of the current Waveform window that is set using the `awvDisplayTitle` function.

```
awvGetWindowTitle(awvGetCurrentWindow())
=> "Transient Analysis"
```

The following example returns the title of the specified Waveform window.

```
awvGetWindowTitle(window(16))
=> "Frequency Response"
```

awvGetXAxisLabel

```
awvGetXAxisLabel(
     w_windowID
     [ ?subwindow x_subwindow ]
     [ ?computed g_computed ]
     )
     => t_label / nil
```

Description

Returns the user-specified or the system-computed label of the x axis.

Arguments

w_windowID	Waveform window ID.
t_label	Label to be displayed for the x axis.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, label for the x axis is set in the current subwindow.
?computed *g_computed*	Specifies whether to return the user-specified label of the x axis. Valid values are: ■ `t`: Returns the system-computed label of the x axis. ■ `nil`: Returns the user-specified label of the x axis.

Value Returned

t_label	Returns the x-axis label in the specified subwindow.
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following example returns the user-specified label for the x axis t in the subwindow `3` of the specified Waveform window.

```
awvGetXAxisLabel(window(15) ?subwindow 3 ?computed nil)
=> "Time in microsecond"
```

awvGetXAxisMajorDivisions

```
awvGetXAxisMajorDivisions(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => x_majorDivisions / nil
```

Description

Returns the number of major divisions that are set on the x axis of a graph in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

x_majorDivisions	Number of the major divisions on the x axis of the specified graph.
nil	The specified Waveform window or subwindow does not exist.

Examples

The following example returns the number of major divisions on the x axis in the current subwindow of the current Waveform window.

```
awvGetXAxisMajorDivisions(awvGetCurrentWindow() ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> 20
```

The following example returns the number of major divisions on the x axis in the subwindow 2 of the specified Waveform window.

```
awvGetXAxisMajorDivisions(window(16) ?subwindow 2)
```

```
=> 30
```

awvGetXAxisMinorDivisions

```
awvGetXAxisMinorDivisions(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => x_minorDivisions / nil
```

Description

Returns the number of minor divisions that are set on the x axis of a graph in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

x_minorDivisions	Number of the minor divisions on the x axis of the specified graph.
nil	The specified Waveform window or subwindow does not exist.

Examples

The following example returns the number of minor divisions on the x axis in the current subwindow of the current Waveform window.

```
awvGetXAxisMinorDivisions(awvGetCurrentWindow() ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> 10
```

The following example returns the number of minor divisions on the x axis in the subwindow 3 of the specified Waveform window.

```
awvGetXAxisMinorDivisions(window(16) ?subwindow 3)
```

```
=> 15
```

awvGetXAxisStepValue

```
awvGetXAxisStepValue(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => x_stepSize / nil
```

Description

Returns the step size for the major divisions of x axis in the specified graph. The step value indicates the spacing between major divisions of x axis.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

x_stepSize	Step size for the major divisions of x axis in the specified graph.
nil	The specified Waveform window or subwindow does not exist.

Examples

The following example returns the step size for the major divisions of x axis in the subwindow 2 of the specified Waveform window.

```
awvGetXAxisStepValue(window(19) ?subwindow 2)
=> 1e-07
```

This indicates that major divisions of x axis are spaced at 100ns (1e-07) in the specified graph.

awvGetXAxisUseStepValue

```
awvGetXAxisUseStepValue(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => t_isStepSizeUsed / nil
```

Description

Indicates whether the step size is used for the major divisions of x axis in the specified graph.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

t_isStepSizeUsed	Indicates whether the step size is used for the major divisions of x axis in the specified graph. Valid values are: ■ `true`: Step size is used. ■ `false`: Step size is not used.
nil	The specified Waveform window or subwindow does not exist.

Examples

The following example indicates that the step size for the major divisions of x axis is used for the graph in the subwindow `2` of the specified Waveform window.

```
awvGetXAxisUseStepValue(window(10) ?subwindow 2)
=> "true"
```

The following example indicates that the step size for the major divisions of x axis is not used for the graph in the subwindow `3` of the specified Waveform window.

```
awvGetXAxisUseStepValue(window(11) ?subwindow 3)
=> "false"
```

awvGetXMarkerNames

```
awvGetXMarkerNames(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => l_XMarkerNames / nil
```

Description

Returns names of all vertical markers in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

l_XMarkerNames	A list containing names of vertical markers that are found in a subwindow of the specified Waveform window.
nil	Either there are no vertical markers found or the specified Waveform window or subwindow does not exist.

Examples

The following example returns names of vertical markers found in subwindow 1 of the specified Waveform window.

```
awvGetXMarkerNames(window(3) ?subwindow 1)
=> ("verticalMarker1" "verticalMarker2" "verticalMarker3")
```

awvGetYAxisLabel

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

Description

Returns the user-specified or the system-computed label of the specified y-axis number.

Arguments

w_windowID	Waveform window ID.
x_yNumber	Number of y axis whose label is to be returned. The value can be any integer from `1` through `4`.

`?subwindow` *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, label for the x axis is set in the current subwindow.

`?computed` *g_computed*

Specifies whether to return the user-specified label of the x axis.

Valid values are:

- `t`: Returns the system-computed label of the x axis.
- `nil`: Returns the user-specified label of the x axis.

`?stripNumber` *x_stripNumber*

Strip number of the trace. If you do not specify this argument, the currently selected strip is used.

If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.

Value Returned

t_label	Returns the y-axis label in the specified subwindow.
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following example returns the user-specified label for the y-axis number `1` in the strip number `2` of subwindow `1` in the current Waveform window.

```
awvGetYAxisLabel(awvGetCurrentWindow() 1 ?computed nil ?stripNumber 2 ?subwindow
1)
=> "Signal Voltage"
```

awvGetYAxisMajorDivisions

```
awvGetYAxisMajorDivisions(
     w_windowID
     x_yNumber
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow x_subwindow ]
     )
     => x_numMajorDivisions / nil
```

Description

Returns the number of major divisions that are set on the scale of the specified y axis of a strip in the specified subwindow of a Waveform window.

Arguments

w_windowID Waveform window ID.

x_yNumber Number of y axis whose number of major divisions is to be set. The value can be any integer from `1` through `4`.

?stripNumber *x_stripNumber*

Strip number of the trace. If you do not specify this argument, the currently selected strip is used.

If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.

?subwindow *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

x_numMajorDivisions

Number of the major divisions set on the scale of the specified y axis in the specified graph.

`nil`	The specified Waveform window, subwindow, y-axis number, or the strip number does not exist.

Examples

The following example returns the number of major divisions set on the y-axis number `1` in the strip number `3` of the current subwindow in the current Waveform window.

```
awvGetYAxisMajorDivisions(awvGetCurrentWindow() 1 ?stripNumber 3 ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
```

The following example returns the number of major divisions set on the y-axis number `1` in the strip number `2` in the subwindow `3` of the specified Waveform window.

```
awvGetYAxisMajorDivisions(window(11) 1 ?stripNumber 2 ?subwindow 3)
=> t
```

awvGetYAxisMinorDivisions

```
awvGetYAxisMinorDivisions(
     w_windowID
     x_yNumber
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow x_subwindow ]
     )
     => x_minorDivisions / nil
```

Description

Returns the number of minor divisions on the scale of the specified y axis of a strip in the specified subwindow of a Waveform window.

Arguments

w_windowID Waveform window ID.

x_yNumber Number of y axis whose number of minor divisions is to be returned. The value can be any integer from `1` through `4`.

?stripNumber *x_stripNumber*

Strip number of the trace. If you do not specify this argument, the currently selected strip is used.

If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.

?subwindow *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

x_minorDivisions Number of the minor divisions on the scale of the specified y axis in the specified graph.

`nil`	The specified Waveform window, subwindow, y-axis number, or the strip number does not exist.

Examples

The following example returns the number of minor divisions on the y-axis number `1` in the strip number `3` of the current subwindow in the current Waveform window.

```
awvGetYAxisMinorDivisions(awvGetCurrentWindow() 1 ?stripNumber 3 ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> 10
```

The following example returns the number of major divisions on the y-axis number `1` in the strip number `2` in the subwindow `3` of the specified Waveform window.

```
awvGetYAxisMinorDivisions(window(11) 1 ?stripNumber 2 ?subwindow 3)
=> 5
```

awvGetYAxisStepValue

```
awvGetYAxisStepValue(
     w_windowID
     x_yNumber
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow x_subwindow ]
     )
     => x_stepSize / nil
```

Description

Returns the step size for the major divisions of specified y axis of a strip in the specified subwindow of a Waveform window. The step value indicates the spacing between major divisions of y axis.

Arguments

w_windowID	Waveform window ID.
x_yNumber	Number of y axis whose step size is to be returned. The value can be any integer from `1` through `4`.
`?stripNumber` *x_stripNumber*	Strip number of the trace. If you do not specify this argument, the currently selected strip is used. If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.
`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

x_stepSize	Step size for the major divisions of the specified y axis in the specified graph.

`nil`	The specified Waveform window, subwindow, y-axis number, or the strip number does not exist.

Examples

The following example returns the step size for the major divisions of the y-axis number `1` of the strip number `2` in the subwindow `3` of the specified Waveform window.

```
awvGetYAxisStepValue(window(19) 1 ?stripNumber 2 ?subwindow 3)
=> 0.0005
```

This indicates that major divisions of the specified y axis are spaced at `500uV` (`0.0005`) in the specified graph.

awvGetYAxisUseStepValue

```
awvGetYAxisUseStepValue(
     w_windowID
     x_yNumber
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow x_subwindow ]
     )
     => t_isStepSizeUsed / nil
```

Description

Indicates whether the step size is used for the major divisions of the specified y axis in the specified graph.

Arguments

w_windowID	Waveform window ID.
x_yNumber	Number of y axis. The value can be any integer from `1` through `4`.

`?stripNumber` *x_stripNumber*

Strip number of the trace. If you do not specify this argument, the currently selected strip is used.

If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.

`?subwindow` *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

t_isStepSizeUsed	Indicates whether the step size is used for the major divisions of y axis in the specified graph. Possible values are: ■ `true`: Step size is used. ■ `false`: Step size is not used.
`nil`	The specified Waveform window, subwindow, y-axis number, or the strip number does not exist.

Examples

The following example indicates that the step size for the major divisions of y-axis number 1 is used for the strip number `2` in the subwindow `3` of the specified Waveform window.

```
awvGetYAxisUseStepValue(window(10) 1 ?stripNumber 2 ?subwindow 3)
=> "true"
```

The following example indicates that the step size for the major divisions of y-axis number 1 is not used for the strip number `3` in the subwindow `2` of the specified Waveform window.

```
awvGetYAxisUseStepValue(window(11) 1 ?stripNumber 3 ?subwindow 2)
=> "false"
```

awvGetYMarkerNames

```
awvGetYMarkerNames(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => l_YMarkerNames / nil
```

Description

Returns names of all horizontal markers in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

l_YMarkerNames	A list containing names of horizontal markers that are found in a subwindow of the specified Waveform window.
nil	Either there are no horizontal markers found or the specified Waveform window or subwindow does not exist.

Examples

The following example returns names of horizontal markers found in subwindow 1 of the specified Waveform window.

```
awvGetYMarkerNames(window(3) ?subwindow 1)
=> ("horizontalMarker1" "horizontalMarker2" "horizontalMarker3")
```

awvInitWindowFunctionAdd

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

Description

Adds a function to the list of initialization 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_function	Function to add to the list. Callback parameter list: *w_windowID*

Value Returned

t	Function is added to the list.
nil	Function cannot be added to the list because of an error.

Examples

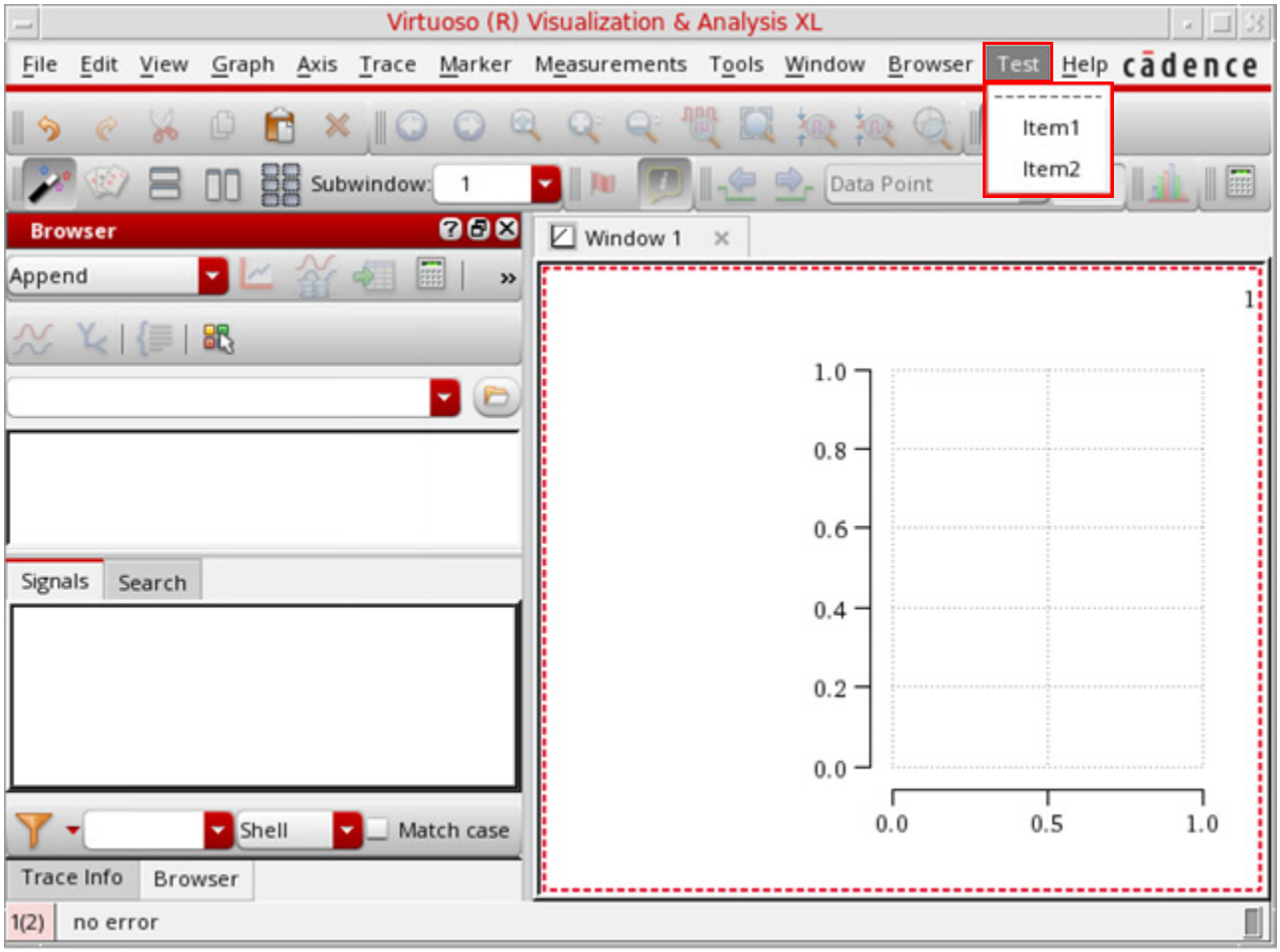
The following example adds a procedure `myCreateMenu`, which creates a new banner menu *Test* in the Results Browser. The *Test* menu has two options, *Item1* and *Item2*.

- When you click *Item1*, the message "`Item 1 Selected`" is printed in the CIW.
- When you click *Item2*, the message "`Item 2 Selected`" is printed in the CIW.

```
myCreateMenu
=> t

windowId=awvGetHiWindow(awvGetCurrentWindow())
procedure(
    myCreateMenu(windowId)
    win=awvGetHiWindow(windowId)
```

```
    let((item1 item2)
        item1=hiCreateMenuItem(
            ?name 'item1
            ?itemText "Item1"
            ?callback "println(\"Item 1 Selected\")"
        )
        item2=hiCreateMenuItem(
            ?name 'item2
            ?itemText "Item2"
            ?callback "println(\"Item 2 Selected\")"
        )
        hiCreatePulldownMenu( 'myMenu "Test" list(item1 item2))
        hiInsertBannerMenu(win myMenu hiGetNumMenus(win))
        )
    )
    awvInitWindowFunctionAdd('myCreateMenu)
=> myCreateMenu
```

```
=> t
```



The following message is printed in CIW when you click *Item1* from the *Test* menu.

```
println("Item 1 Selected")
"Item 1 Selected"
```

The following message is printed in CIW when you click *Item2* from the *Test* menu.

```
println("Item 2 Selected")
"Item 2 Selected"
```

awvInitWindowFunctionDel

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

Description

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

Arguments

u_function	Function to be deleted from the list of functions that are called when a new Waveform window is opened.

Value Returned

`t`	Function is deleted from the list.
`nil`	Function cannot be deleted from the list because of an error.

Examples

The following example adds a procedure `myCreateMenu`, which creates a new banner menu *Test* in the Results Browser. The *Test* menu has two options, *Item1* and *Item2*.

- When you click *Item1*, the message "`Item 1 Selected`" is printed in the CIW.
- When you click *Item2*, the message "`Item 2 Selected`" is printed in the CIW.

```
myCreateMenu
=> t

windowId=awvGetHiWindow(awvGetCurrentWindow())
procedure(
    myCreateMenu(windowId)
    win=awvGetHiWindow(windowId)
    let((item1 item2)
        item1=hiCreateMenuItem(
            ?name 'item1
```

```
            ?itemText "Item1"
            ?callback "println(\"Item 1 Selected\")"
        )
        item2=hiCreateMenuItem(
            ?name 'item2
            ?itemText "Item2"
            ?callback "println(\"Item 2 Selected\")"
        )
        hiCreatePulldownMenu( 'myMenu "Test" list(item1 item2))
        hiInsertBannerMenu(win myMenu hiGetNumMenus(win))
        )
    )
    awvInitWindowFunctionAdd('myCreateMenu)
=> myCreateMenu
=> t
```

The following example returns a list of names of the initialization functions for Waveform Windows.

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

The following example deletes the initialization function `myCreateMenu` from the list of functions that are called when a new Waveform window is opened.

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

Now, the function `awvInitWindowFunctionGet` returns `nil` because the initialization function `myCreateMenu` has been deleted.

```
awvInitWindowFunctionGet()
=> nil
```

awvInitWindowFunctionGet

```
awvInitWindowFunctionGet(
     )
     => l_initFunctionList / nil
```

Description

Returns a list of names of initialization functions that are currently added to the list of functions that are called when a Waveform window is opened.

This list is empty by default.

Arguments

None

Value Returned

l_initFunctionList	A list containing names of initialization functions.
nil	There are no initialization functions.

Examples

The following example adds a procedure `myCreateMenu`, which creates a new banner menu *Test* in the Results Browser. The *Test* menu has two options, *Item1* and *Item2*.

- When you click *Item1*, the message "`Item 1 Selected`" is printed in the CIW.
- When you click *Item2*, the message "`Item 2 Selected`" is printed in the CIW.

```
myCreateMenu
=> t

windowId=awvGetHiWindow(awvGetCurrentWindow())
procedure(
    myCreateMenu(windowId)
    win=awvGetHiWindow(windowId)
    let((item1 item2)
        item1=hiCreateMenuItem(
```

```
            ?name 'item1
            ?itemText "Item1"
            ?callback "println(\"Item 1 Selected\")"
        )
        item2=hiCreateMenuItem(
            ?name 'item2
            ?itemText "Item2"
            ?callback "println(\"Item 2 Selected\")"
        )
        hiCreatePulldownMenu( 'myMenu "Test" list(item1 item2))
        hiInsertBannerMenu(win myMenu hiGetNumMenus(win))
        )
    )
    awvInitWindowFunctionAdd('myCreateMenu)
=> myCreateMenu
=> t
```

The following example returns a list of names of the initialization functions for Waveform Windows.

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

The following example deletes the initialization function `myCreateMenu` from the list of functions that are called when a new Waveform window is opened.

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

Now, the function `awvInitWindowFunctionGet` returns `nil` because the initialization function `myCreateMenu` has been deleted.

```
awvInitWindowFunctionGet()
=> nil
```

awvIsPlotWindow

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

Description

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

Arguments

w_windowID	ID of the window.

Value Returned

`t`	The specified window is a Waveform window.
`nil`	The specified window is not a Waveform window.

Examples

The following example indicates that the specified window is a Waveform window.

```
awvIsPlotWindow(window(7))
=> t
```

awvLoadCustomCalcFunction

```
awvLoadCustomCalcFunction(
     [ ?funcList l_funcList ]
     [ ?fileName t_fileName ]
     [ ?templateFileName t_templateFileName ]
     [ @rest l_args ]
     )
     => t / nil
```

Description

Adds the functions specified in the SKILL file (`.il`) and UI template file (`.ocn`) to the *Custom Functions* drop-down list in the *Function Panel* of Virtuoso Visualization and Analysis XL Calculator.

The SKILL file contains the definition of the custom calculator functions. The UI template file (`.ocn`) contains the user-interface template of the custom calculator functions.

The user-interface template can be obtained either from the SKILL file or from a separate `.ocn` file.

Arguments

`?funcList` *`l_funcList`*

List containing names of the custom functions defined in the SKILL file that you want to add to the *Custom Functions* drop-down list in the *Function Panel* of Calculator.

`?fileName` *`t_fileName`*

Path to the SKILL file (`.il`) that contains SKILL definitions of custom functions. This SKILL file may also contain the UI templates for custom functions.

If the SKILL file is saved in the current working directory, you can specify only the file name.

`?templateFileName` *`t_templateFileName`*

Path to the `.ocn` file that contains the UI templates for custom functions.

If the `.ocn` file is saved in the current working directory, you can specify only the file name.

`@rest` *`l_args`*	List of additional arguments passed to the function.

Value Returned

`t`	Custom functions are added to Function Panel successfully.
`nil`	Custom functions cannot be added to Function Panel because of an error.

Examples

The following example adds a custom function `customBandwidth` to the *Custom Functions* drop-down list in the *Function Panel*. The SKILL definition of the custom function is saved in the `myCustomCalFunction.il` and the UI template of the custom function is saved in a separate `myCustomCalFunction.ocn` file.

```
awvLoadCustomCalcFunction(?funcList "customBandwidth" ?fileName "/home/user/
myCustomCalFunction.il" ?templateFileName "/home/user/myCustomCalFunction.ocn")
=> t
```

The following example adds two custom functions `customFrequency` and `customWavelength` to the *Custom Functions* drop-down list in the *Function Panel*.

The SKILL definitions and UI templates for both custom functions are saved in a single SKILL file `myCustomSKILL.il`.

```
awvLoadCustomCalcFunction(?funcList list("customFrequency" "customWavelength")
?fileName "/home/user/myCustomSKILL.il")
=> t
```

awvLoadEyeMask

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

Description

Loads the eye mask saved in the specified 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 in the *Eye Mask* tab of the Eye Diagram assistant.

Arguments

`?fileName` *`t_fileName`*

Path to the VCSV file from which you want to load the eye mask.

Value Returned

`t`	The eye mask is loaded successfully.
`nil`	The eye mask cannot be loaded because of an error.

Examples

The following example loads the eye mask saved in the eyeMask.vscv file.

```
awvLoadEyeMask(?fileName "/servers/user/design/eyeMask.vcsv")
=> t
```

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`	Load menu is successfully displayed.
`nil`	Load menu cannot be displayed because of an error.

Examples

The following example shows how to run the awvLoadMenu function.

```
awvLoadMenuCB()
=> nil
```

awvLoadSharedCustomFunctionsFile

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

Description

Shares definitions and UI templates for custom functions among multiple users from a central file location.

Arguments

t_centralFileName	Name of the configuration file (.ini), containing definitions and UI templates for custom functions, that you want to share.

Value Returned

`t`	Definitions and UI templates for custom functions is shared successfully.
`nil`	Definitions and UI templates for custom functions cannot be shared because of an error.

Examples

The following example shows how to share definitions and UI templates for custom functions defined in a configuration file (`.ini`) from a central location.

1. Create a configuration file named `central.ini`.
2. Specify functions to add three custom functions `customFunction1`, `customFunction2`, and `customFunction3` in the file `central.ini`.
   - `awvLoadCustomCalcFunction(?funcList "customFunction1" ?fileName "myCustomCalFunction.il" ?templateFileName "myCustomCalFunction.ocn")`
   - `awvLoadCustomCalcFunction(?funcList list("customFunction2" "customFunction3") ?fileName "skill.il")`

3. In the `.cdsinit` file, add the `awvLoadSharedCustomFunctionsFile` function to share and load the custom functions.

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

   You can verify that shared custom functions are loaded successfully by running the following in CIW.

   ```
   awvLoadSharedCustomFunctionsFile("/home/user/central.ini")
   => t
   ```

   The shared custom functions appear in red in Function Panel.

awvLoadWindow

```
awvLoadWindow(
     w_windowID
     t_fileName
     [ ?resultsDir t_resultsDirectory ]
     )
     => t / nil
```

Description

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

You can save the state of a Waveform window to an XML file with the `.grf` extension by using the `awvSaveWindow` function.

Arguments

w_windowID	ID of the Waveform window in which you want to load the state.
t_fileName	Path or name of the file in which the state of the specified Waveform window is stored. You must have read permission to this file.
`?resultsDir` *t_resultsDirectory*	Directory containing the PSF files.

Value Returned

`t`	Waveform window is initialized with the specified state file.
`nil`	Waveform window cannot be initialized because of an error.

Examples

The following example saves the state of the Waveform window `7` to `myFile.grf` in the `/home/user` directory.

```
awvSaveWindow(window(7) "/home/user/myFile")
=> t
```

The following examples initialize the state of Waveform window `8` from the information saved in the `myFile.grf`.

```
awvLoadWindow(window(8) "/home/user/myFile.grf")
=> t
awvLoadWindow(window(8) "/home/user/myFile.grf" ?resultsDir "./simulation/
maestro/results/maestro/ExplorerRun.0/psf/AC/")
=> t
```

awvLogXAxis

```
awvLogXAxis(
     w_windowID
     g_state
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Specifies whether to set the display of x axis to logarithmic in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
g_state	Specifies whether to set the display of x axis to logarithmic or linear. Valid values are: ■ `t`: Sets the display of x axis to logarithmic. ■ `nil`: Sets the display of x axis to linear. This argument will not take effect if the display mode is `smith`.
`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, label for the x axis is set in the current subwindow.

Value Returned

`t`	Display of the x axis is set successfully.
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following example sets the display of the x axis to logarithmic in the subwindow `4` of the specified Waveform window.

```
awvLogXAxis(window(11) t ?subwindow 4)
=> t
```

The following example sets the display of the x axis to linear in the subwindow `5` of the specified Waveform window.

```
awvLogXAxis(window(12) nil ?subwindow 5)
=> t
```

awvLogYAxis

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

Description

Specifies whether to set the display of y axis to logarithmic in a subwindow of the specified Waveform window.

Arguments

w_windowID — Waveform window ID.

x_yNumber — Number of y axis on which logarithmic scale is to be set. The value can be any integer from `1` through `4`.

g_state — Specifies whether to set the display of y axis to logarithmic or linear.

Valid values are:

- `t`: Sets the display of y axis to logarithmic.
- `nil`: Sets the display of y axis to linear.

`?stripNumber` *x_stripNumber*

Strip number of the trace. If you do not specify this argument, the currently selected strip is used.

If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.

`?subwindow` *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, label for the x axis is set in the current subwindow.

Value Returned

`t`	Display of the y axis is set successfully.
`nil`	The specified Waveform window, subwindow, y-axis number, or the strip number does not exist.

Examples

The following example sets the display for the y-axis number `2` of the strip `3` to logarithmic in subwindow `4` of the specified Waveform window.

```
awvLogYAxis(window(11) 2 t ?stripNumber 3 ?subwindow 4)
=> t
```

The following example sets the display for the y-axis number `3` of the strip `3` to linear in subwindow `4` of the specified Waveform window.

```
awvLogYAxis(window(11) 3 nil ?stripNumber 3 ?subwindow 4)
=> t
```

awvPlaceAMarker

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

Description

Places a marker of type A at the specified location on the specified trace.

Arguments

w_windowID	Waveform window ID
x_traceIndex	Index number that identifies the trace. This is an integer value.
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*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.
?positionMode *t_positionMode*	Specifies the position mode of the marker. Valid values are: ■ x: Marker is placed at the specified *xLoc* value and the nearest *yLoc* value on the trace. This is the default value. ■ y: Marker is placed at the specified *yLoc* value and the nearest *xLoc* value on the trace. ■ xy: Marker is placed at the specified *xLoc* and *yLoc* values on the trace.

Value Returned

t_markerID	Returns the ID of the marker after the marker is placed on the specified trace.
nil	The marker cannot be placed on the trace because of an error.

Examples

The following example creates a Waveform window.

```
win=awvCreatePlotWindow()
=> window:10
```

Opens simulation results stored in the specified directory:

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

Plots the waveforms of net10 and out signals from the simulation results:

```
awvPlotWaveform(
                win
                list(v("net10" ?result "tran-tran") v("out" ?result "tran-tran"))
                )
=> t
```

Returns the index numbers and the signal names plotted in the Waveform window:

```
awvGetWaveNameList(win)
=>
((1 2)
    ("net10" "out")
)
```

Places a marker at x=100n on the trace out in the subwindow 1 of the specified waveform window win= window:10.

```
awvPlaceAMarker(win 2 100n 0 ?subwindow 1)
=> "refPointMarker[50.99.2]"
```

Places a marker at x=200n on the trace out in the current subwindow of the specified Waveform window:

```
awvPlaceAMarker(win 2 200n 0 ?positionMode "x")
=> "refPointMarker[50.99.2]"
```

Places a marker at y=3 on the trace net10in the current subwindow of the specified Waveform window:

```
awvPlaceAMarker(win 1 200n 3 ?positionMode "y")
=> "refPointMarker[50.99.2]"
```

Places a marker at `x=50n` and `y=2.5` on the trace `out` in the subwindow `1` of the specified Waveform window:

```
awvPlaceAMarker(win 2 50n 2.5 ?subwindow 1 ?positionMode "xy")
=> "refPointMarker[50.99.2]"
```

awvPlaceBMarker

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

Description

Places a marker of type B at the specified location on the specified trace.

Arguments

w_windowID	Waveform window ID
x_traceIndex	Index number that identifies the trace. This is an integer value.
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*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.
?positionMode *t_positionMode*	Specifies the position mode of the marker. Valid values are: ■ x: Marker is placed at the specified *xLoc* value and the nearest *yLoc* value on the trace. This is the default value. ■ y: Marker is placed at the specified *yLoc* value and the nearest *xLoc* value on the trace. ■ xy: Marker is placed at the specified *xLoc* and *yLoc* values on the trace.

Value Returned

t_markerID	Returns the ID of the marker after the marker is placed on the specified trace.
nil	The marker cannot be placed on the trace because of an error.

Examples

The following example creates a Waveform window.

```
win=awvCreatePlotWindow()
=> window:10
```

Opens simulation results stored in the specified directory:

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

Plots the waveforms of net10 and out signals from the simulation results.

```
awvPlotWaveform(
                win
                list(v("net10" ?result "tran-tran") v("out" ?result "tran-tran"))
                )
=> t
```

Returns the index numbers and the signal names plotted in the Waveform window:

```
awvGetWaveNameList(win)
=>
((1 2)
    ("net10" "out")
)
```

Places a marker at x=100n on the trace out in the subwindow 1 of the specified waveform window win= window:10.

```
awvPlaceBMarker(win 2 100n 0 ?subwindow 1)
=> "refPointMarker[50.99.4]"
```

Places a marker at x=200n on the trace out in the current subwindow of the specified Waveform window:

```
awvPlaceBMarker(win 2 200n 0 ?positionMode "x")
=> "refPointMarker[50.99.4]"
```

Places a marker at y=3 on the trace net10in the current subwindow of the specified Waveform window:

```
awvPlaceBMarker(win 1 200n 3 ?positionMode "y")
=> "refPointMarker[50.99.4]"
```

Places a marker at `x=50n` and `y=0.5` on the trace `net10` in the subwindow `1` of the specified Waveform window:

```
awvPlaceBMarker(win 1 50n 0.5 ?subwindow 1 ?positionMode "xy")
=> "refPointMarker[50.99.4]"
```

awvPlaceBookmark

```
awvPlaceBookmark(
     w_windowID
     t_bookmarkType
     l_location
     [ ?waveIndex x_waveIndex ]
     [ ?parent t_parentGroup ]
     [ ?description t_description ]
     [ ?visible g_visible ]
     [ ?properties l_propertiesList ]
     [ ?subwindow x_subwindow ]
     )
     => l_bookmarkID / nil
```

Description

Adds a bookmark of the specified type in a subwindow of the specified Waveform window.

The bookmarks can be organized into groups, where a bookmark group can contain another group.

Arguments

w_windowID	Waveform window ID.
t_bookmarkType	Type of the bookmark to be created. Valid values are: ■ `xrange`: Creates an X Range bookmark ■ `yrange`: Creates a Y Range bookmark ■ `region`: Creates a Region bookmark ■ `point`: Creates a point bookmark

l_location

A list of two waveform coordinates that specify the location of an `xrange`, `yrange`, or a `point` bookmark.

A list of four waveform coordinates that specify the location of a `region` bookmark.

A few examples are:

- `xrange`: `list(25n 125n)`
- `yrange`: `list(0.75 3)`
- `point`:`list(25n 0.75)`
- `region`: `list(25n 0.75 125n 3)`

`?waveIndex` *x_waveIndex*

Integer identifying the waveform curve in which bookmark is to be created.

`?parent` *t_parentGroup*

Name of the parent group when the bookmark is organized into a group or nested groups.

`?description` *t_description*

Description to be added to the bookmark.

`?visible` *g_visible*

Specifies whether the bookmark is visible by default.

Valid values are:

- `t`: Bookmark is visible.
- `nil`: Bookmark is not visible.

Default value is `t`.

`?properties` *l_propertiesList*

List of properties that can be set to the bookmark.

For example:

- `fillColor`: Color to be filled in the bookmark. Default value is `blue`.
- `font`: Font of the bookmark.
- `lineStyle`: Line style of the bookmark. Valid values are `solid`, `dash`, `dot`, `dashDot`, and `dashDotDot`. For a `region` bookmark, default value is `solid`. For `xrange` and `yrange` bookmarks, default value is `dash`.
- `foreground`: Foreground color of the bookmark. Default value is `lightblue`.
- `notation`: Notation of the bookmark. Valid values are `suffix`, `scientific`, and `engineering`. Default value is `suffix`.
- `showlabel`: Specifies when to show the bookmark label. Valid values are `off`, `on`, `on when hover`. Default value is `on when hover`.
- `sigDigitsMode`: Mode of significant digits. Valid values are `auto` and `manual`. Default value is `auto`.
- `significantDigits`: Number of significant digits. Default value is `4`.
- `zoomScaleFactor`: Zoom scale factor. Default value is `1.5`.
- `defaultLabel`: Adds a string to the default label of the bookmark. Default value is `BM`*`bookmarkNumber`*. For example, `BM2`.

`?subwindow` *`x_subwindow`*

Number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

l_bookmarkID	Identification number of the bookmark.
nil	Bookmark cannot be added because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
win=awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

The following example lists results that are available in the currently open results directory.

```
results()
=> tran(tranOp ac dcOp dcOpInfo model)
```

The following example selects the tran result from the current results directory. The tran result contains signal net10 and out.

```
selectResult('tran)
=> stdobj@0x32253b30
```

The following example creates a waveform object `signal1`, representing the waveform of the signal `net10` that is available in the `tran` result.

```
signal1=v("net10")
=> srrWave:0x36552020
```

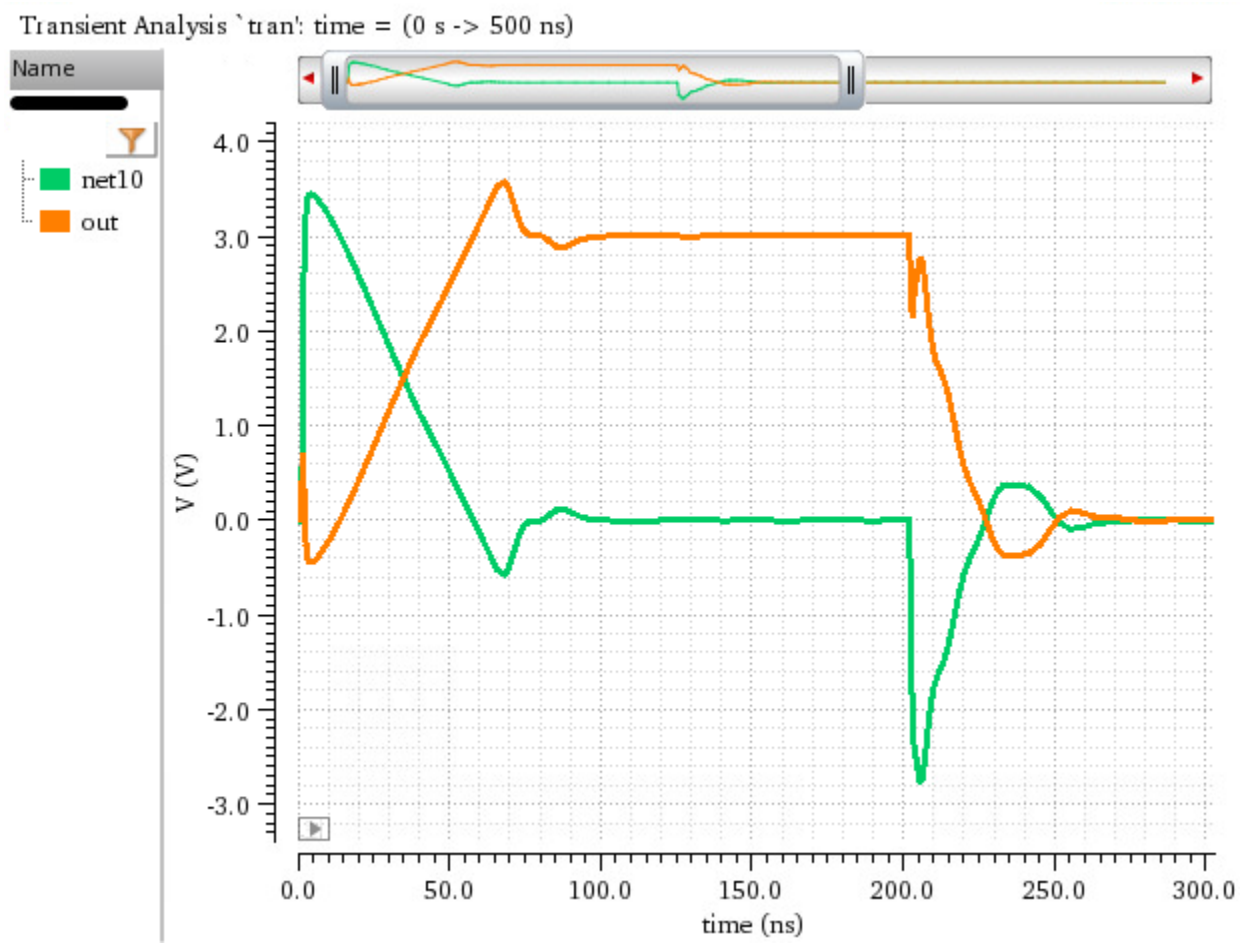
The following example creates a waveform object `signal2`, representing the waveform of the signal `out` that is available in the `tran` result.

```
signal2=v("out")
=> srrWave:0x36552030
```

The following example plots the waveforms of signals net10 and out in the Waveform window `win`.

```
awvPlotWaveform(
                win
                list(signal1 signal2)
                ?expr list("net10" "out")
                ?color list("y18" "y6")
```

```
                ?lineStyle list("solid" "solid")
                ?lineThickness list("thick" "thick")
                )
=> t
```

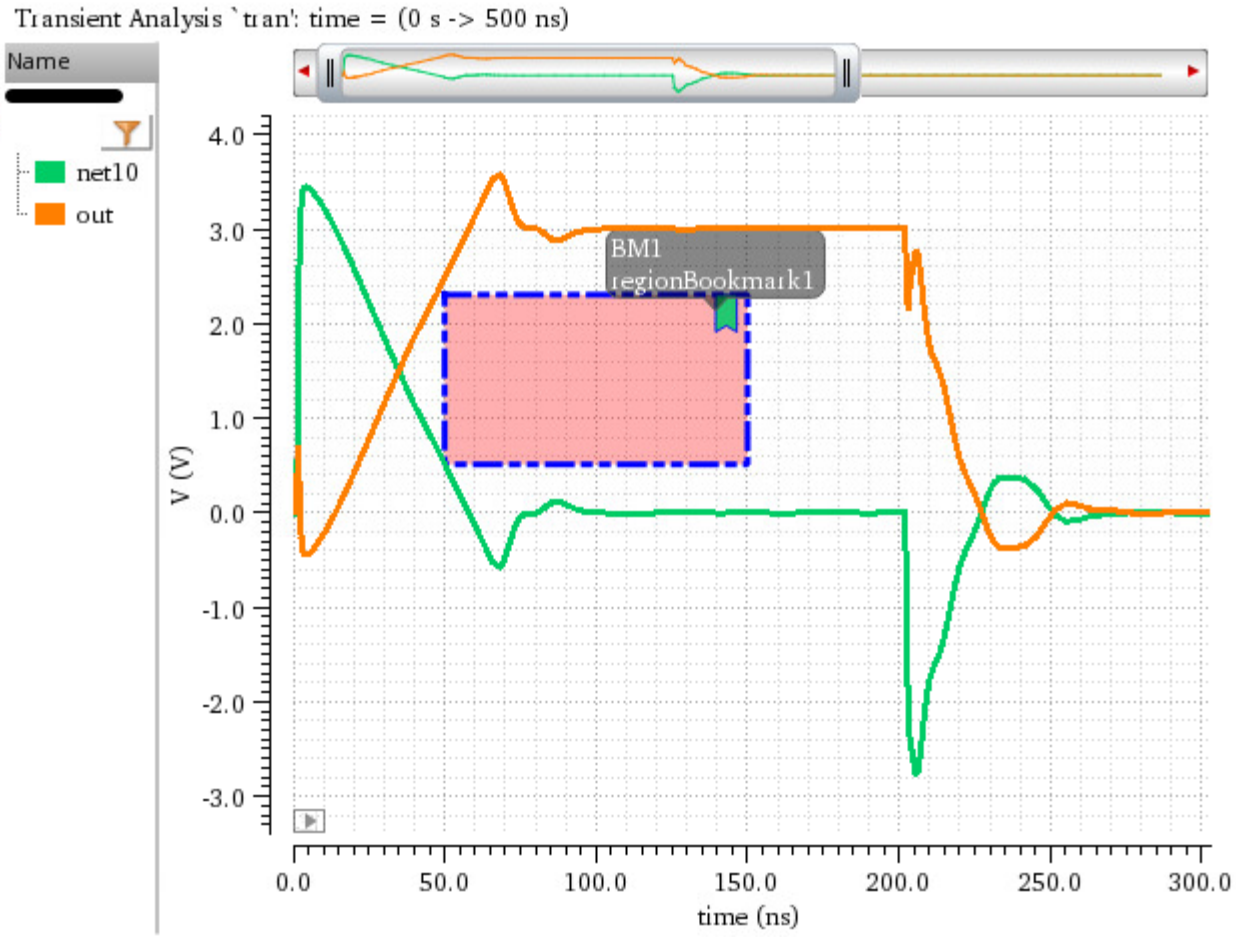


The following example returns the wave index numbers and the signal names plotted in the Waveform window, *win*.

```
awvGetWaveNameList(win)
=>
((1 2)
    ("net10" "out")
)
```

The following example creates a region bookmark with the specified properties at the specified location. Note that the region bookmark `bm1` is attached to the trace `net10` because it has the index number `1`.

```
bm1=awvPlaceBookmark(win "region" list(50n 0.5 150n 2.3) ?waveIndex 1 ?description
"regionBookmark1" ?properties list(list("fillColor" "red") list("foreground"
"blue") list("lineStyle" "dashDotDot")))
=> ("regionBookmark[1.1.1]")
```

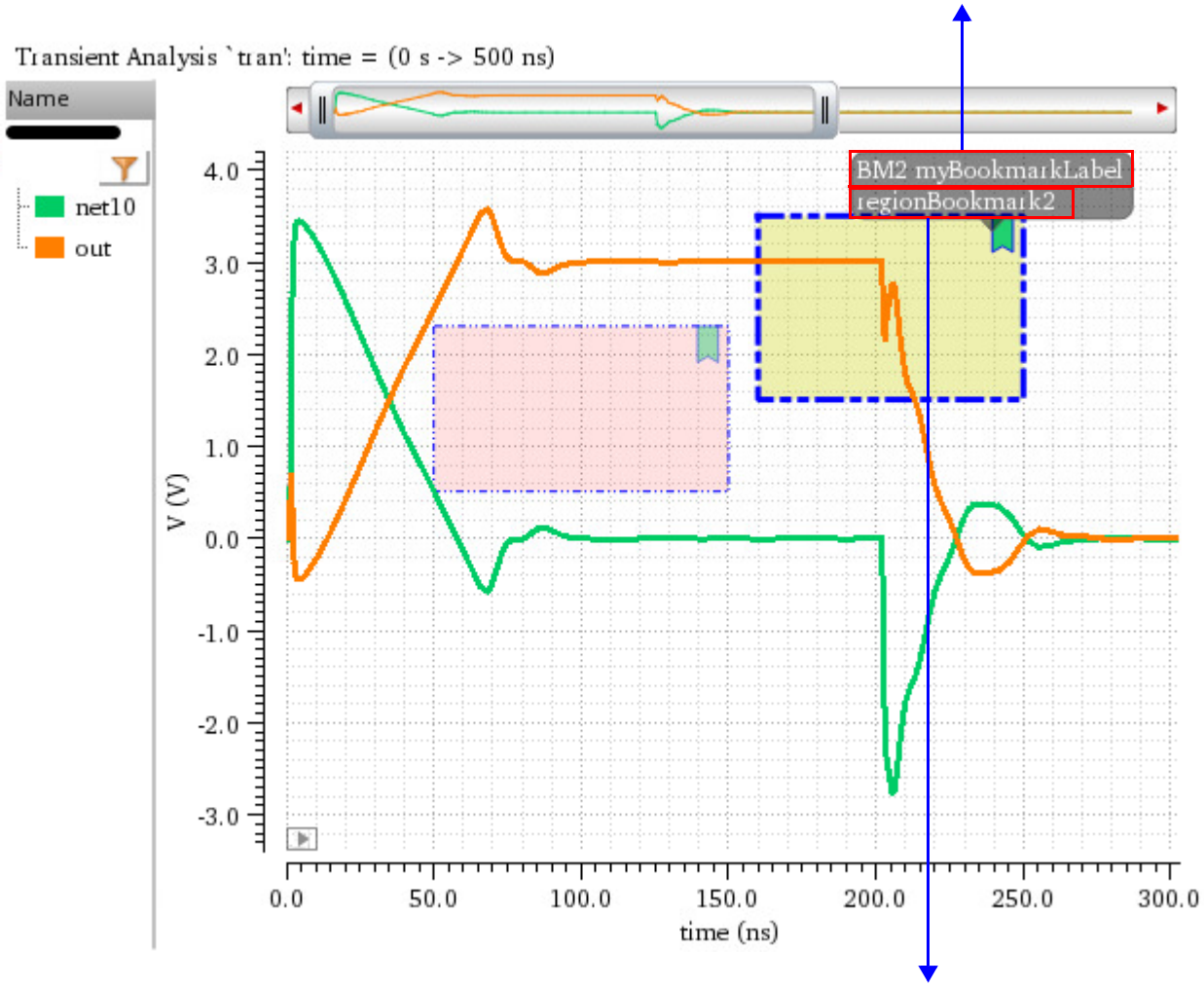


The following example creates another region bookmark with the specified properties at the specified location. The region bookmark `bm2` is attached to the trace `out` because it has the wave index number `2`.

```
bm2=awvPlaceBookmark(win "region" list(160n 1.5 250n 3.5) ?waveIndex 2 ?description
"regionBookmark2" ?properties list(list("fillColor" "yellow") list("defaultLabel"
"myBookmarkLabel") list("foreground" "blue") list("lineStyle" "dashDotDot")))
```

```
=> ("regionBookmark[1.1.2]")
```

A string `myBookmarkLabel` added to the default label `BM2`



Description of the region bookmark `BM2`

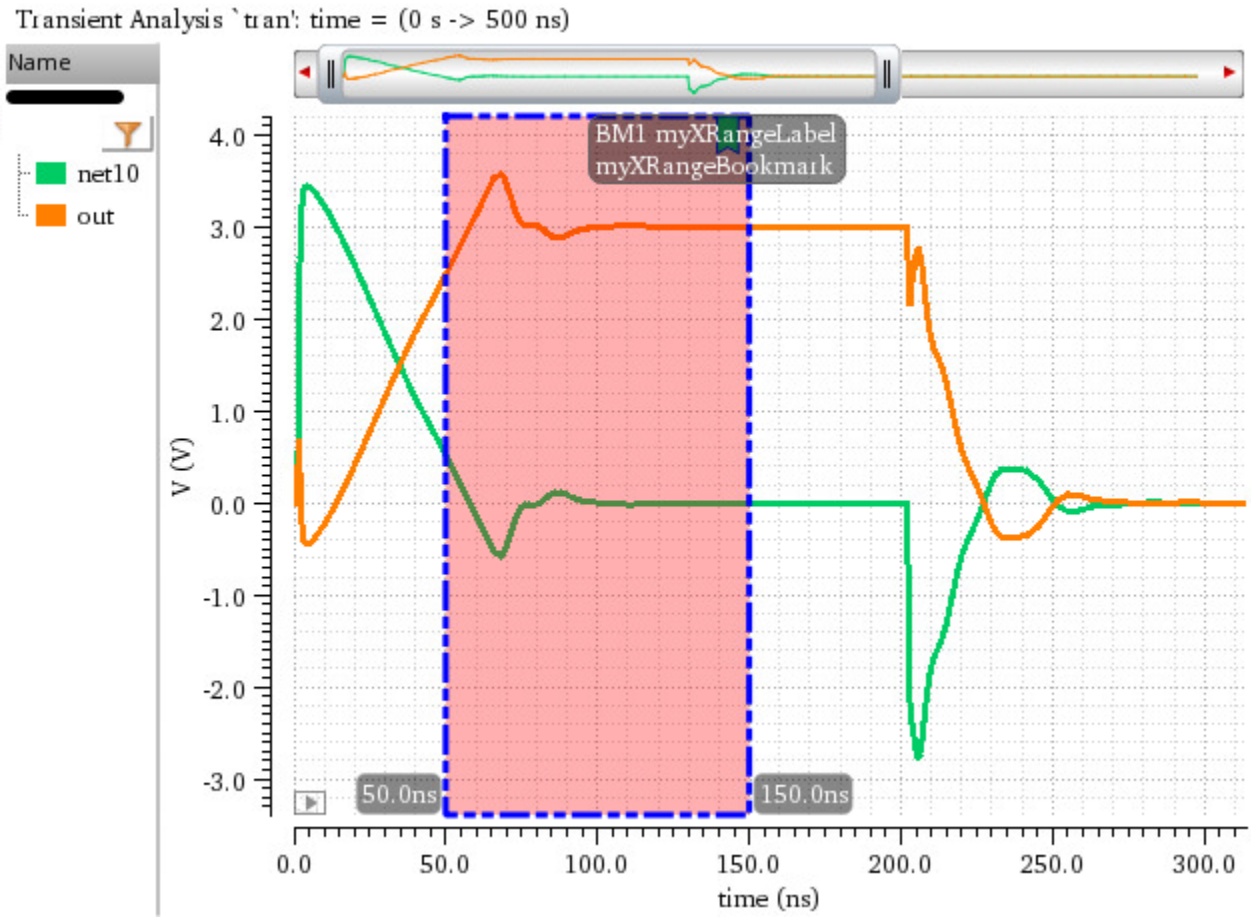
Note that a string `myBookMarkLabel` is also added to the default label of the bookmark `BM2`.

The following examples delete bookmarks `bm1` and `bm2` from the specified Waveform window.

```
awvDeleteMarker(win bm1)
=> t
awvDeleteMarker(win bm2)
=> t
```

The following example creates an `xrange` bookmark with the specified properties at the specified location. Note that the xrange bookmark `bm1` is attached to the trace `net10` because it has the index number `1`.

```
bm1=awvPlaceBookmark(win "xrange" list(50n 150n) ?waveIndex 1 ?description
"myXRangeBookmark" ?properties list(list("fillColor" "red") list("foreground"
"blue") list("defaultLabel" "myXRangeLabel") list("lineStyle" "dashDotDot")))
=> ("xrangeBookmark[1.1.1]")
```

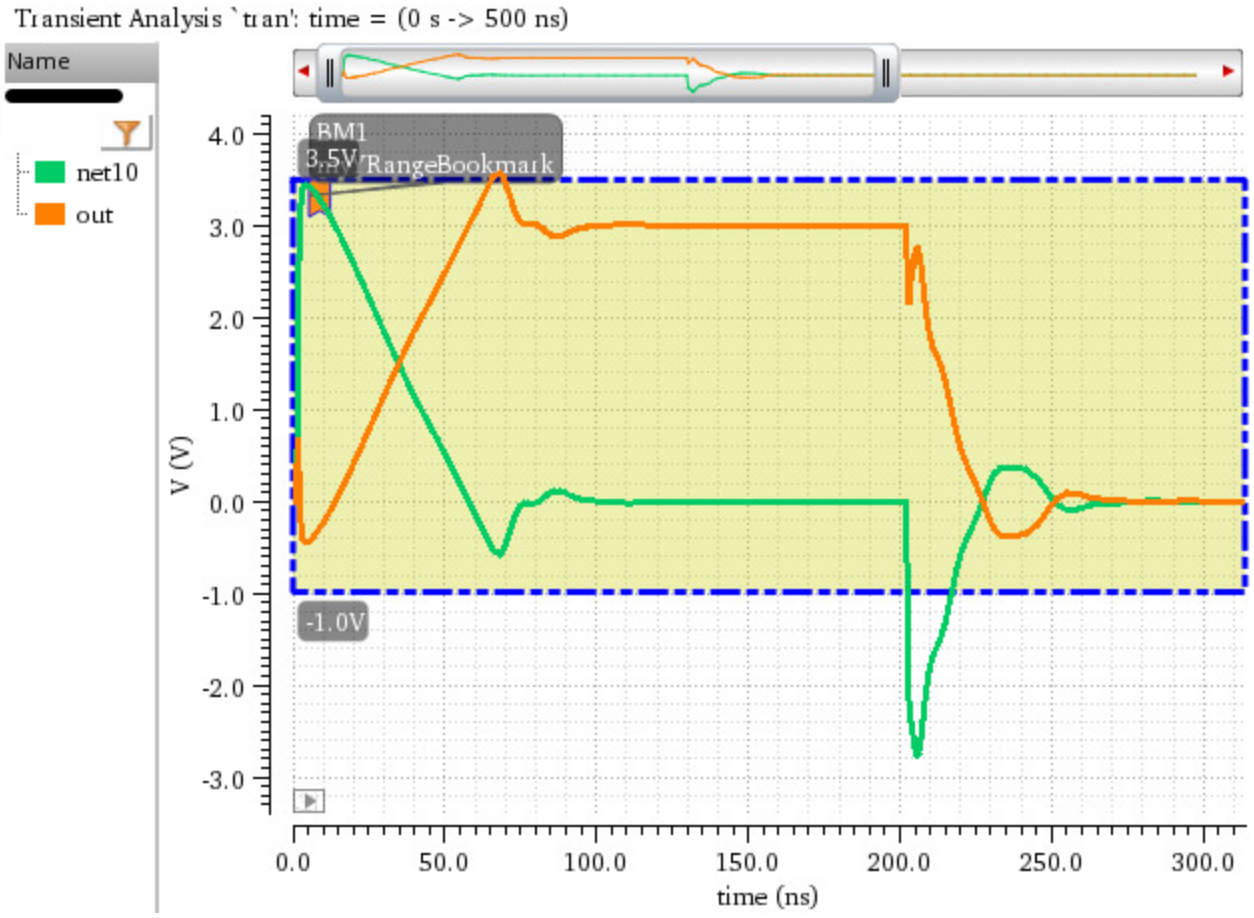


The following example deletes the `xrange` bookmark `bm1` from the specified Waveform window.

```
awvDeleteMarker(win bm1)
=> t
```

The following example creates a `yrange` bookmark with the specified properties at the specified location. Note that the yrange bookmark `bm1` is attached to the trace `out` because it has the index number `2`.

```
bm1=awvPlaceBookmark(win "yrange" list(-1 3.5) ?waveIndex 2 ?description
"myYRangeBookmark" ?properties list(list("fillColor" "yellow") list("foreground"
"blue") list("lineStyle" "dashDotDot")))
=> ("yrangeBookmark[1.1.1]")
```



awvPlaceCircleMarker

```
awvPlaceCircleMarker(
     w_windowID
     n_xLoc
     n_yLoc
     n_radius
     [ ?subwindow x_subwindow ]
     [ ?label t_label ]
     [ ?startAngle n_startAngle ]
     [ ?spanAngle n_spanAngle ]
     [ ?outlineColor t_outlineColor ]
     [ ?fillColor t_fillColor ]
     )
     => t_markerID / nil
```

Description

Places a circular marker at the specified location on a circular graph.

Arguments

w_windowID	Waveform window ID
n_xLoc	The x coordinate of the center of the circular marker.
n_yLoc	The y coordinate of the center of the circular marker.
n_radius	The radius (magnitude) of the circular marker.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.
?label *t_label*	Label to be displayed on the circular marker.
?startAngle *n_startAngle*	The start angle of the pie slice in degree or radian.
?spanAngle *n_spanAngle*	The end angle of the pie slice in degree or radian.
?outlineColor *t_outlineColor*	

Outline color of the circular marker.

`?fillColor` *`t_fillColor`*

Color to be filled in the circular marker.

Value Returned

`t_markerID`	Returns the ID of the circular marker after the marker is placed on the specified circular graph.
`nil`	The point marker cannot be placed on the trace because of an error.

Examples

The following example places a circular marker `cM1` on the circular graph in the current Waveform window. The center of the circular marker is located at `x=0.04` (`40.0m`) and `y=0.05` (`50.0m`). The radius of the circular marker is `0.5`. The outline color of the circular marker is `red`. The circular marker is filled with `yellow` color.

```
awvPlaceCircleMarker(awvGetCurrentWindow() 0.04 0.05 0.5 ?label "%F (%C:%R)"
     ?outlineColor "red" ?fillColor "yellow")

=> "circleMarker[51.17.1]"
```

The following example creates a Waveform window.

```
win=awvCreatePlotWindow()

=> window:132
```

Opens simulation results stored in the specified directory:

```
openResults("./simulation/opamp090/full_diff_opamp/maestro/results/maestro/
     Interactive.13/psf/AC/psf")

=> "./simulation/opamp090/full_diff_opamp/maestro/results/maestro/Interactive.13/
     psf/AC/psf"
```

Sets the display mode of the current Waveform window to `smith`:

```
awvSetDisplayMode(win "smith")

=> t
```

Plots the signal `OUT` in a circular graph:

```
awvPlotWaveform(
                win
```

```
                list(v("OUT" ?result "ac-ac"))
                )
=> t
```

Places a circular marker `cM2` at the specified location on the circular graph in the current Waveform window:

```
awvPlaceCircleMarker(win
                    0.04
                    0.05
                    0.5
                    ?label "%F (%C:%R)"
                    ?outlineColor "red"
                    ?fillColor "yellow"
                    )
=> "circleMarker[52.18.2]"
```

awvPlacePointMarker

```
awvPlacePointMarker(
     w_windowID
     x_traceIndex
     n_xLoc
     n_yLoc
     [ ?subwindow x_subwindow ]
     [ ?positionMode t_positionMode ]
     )
     => t_markerID / nil
```

Description

Places a point marker at the specified location on the specified trace.

Arguments

w_windowID	Waveform window ID
x_traceIndex	Index number that identifies the trace. This is an integer value.
n_xLoc	The x coordinate at which the point marker is to be placed.
n_yLoc	The y coordinate at which the point marker is to be placed.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.
?positionMode *t_positionMode*	Specifies position mode of the point marker. Valid values are: ■ x: The point marker is placed at the specified *xLoc* value and the nearest *yLoc* value on the trace. This is the default value. ■ y: The point marker is placed at the specified *yLoc* value and the nearest *xLoc* value on the trace. ■ xy: The point marker is placed at the specified *xLoc* and *yLoc* values on the trace.

Value Returned

t_markerID	Returns the ID of the point marker after the marker is placed on the specified trace.
nil	The point marker cannot be placed on the trace because of an error.

Examples

The following example creates a Waveform window.

```
win=awvCreatePlotWindow()
=> window:10
```

Opens simulation results stored in the specified directory:

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

Plots the waveforms of `net10` and `out` signals from the simulation results.

```
awvPlotWaveform(
                win
                list(v("net10" ?result "tran-tran") v("out" ?result "tran-tran"))
                )
=> t
```

Returns the index numbers and the signal names plotted in the Waveform window:

```
awvGetWaveNameList(win)
=>
((1 2)
    ("net10" "out")
)
```

Places a point marker at `x=200n` on the trace `out` in the subwindow `1` of the specified Waveform window:

```
awvPlacePointMarker(win 2 200n 2.5 ?subwindow 1 ?positionMode "x")
=> "pointMarker[50.99.6]"
```

Places a point marker at `y=1.5` on the trace `net10` in current subwindow of the specified Waveform window:

```
awvPlacePointMarker(win 1 50n 1.5 ?positionMode "y")
=> "pointMarker[50.99.7]"
```

Places a point marker at `x=50n` and `y=2.5` on the trace `out` in the subwindow `1` of the specified Waveform window:

```
awvPlacePointMarker(win 2 50n 2.5 ?subwindow 1 ?positionMode "xy")
=> "pointMarker[50.99.8]"
```

awvPlaceQContour

```
awvPlaceQContour(
     w_windowID
     n_QValue
     [ ?subwindow x_subwindow ]
     [ ?label t_label ]
     [ ?outlineColor t_outlineColor ]
     [ ?fillColor t_fillColor ]
     )
     => t_markerID / nil
```

Description

Adds two Q contour curves to the specified circular graph. Q is defined as `Reactance/ Resistance`. A Q contour curve shows the curve of constant Q value on the Impedance Smith Chart.

Arguments

w_windowID	Waveform window ID
n_QValue	Q value for which the Q contour curves are to be drawn.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.
?outlineColor *t_outlineColor*	Outline color of the Q contour curves.
?fillColor *t_fillColor*	Color to be filled in the Q contour curves.

Value Returned

t_markerID	Returns IDs of two Q contour curves added for the specified Q value.
nil	Q contour curves cannot be added because of an error.

Examples

The following function creates a Waveform window and returns ID of the Waveform window.

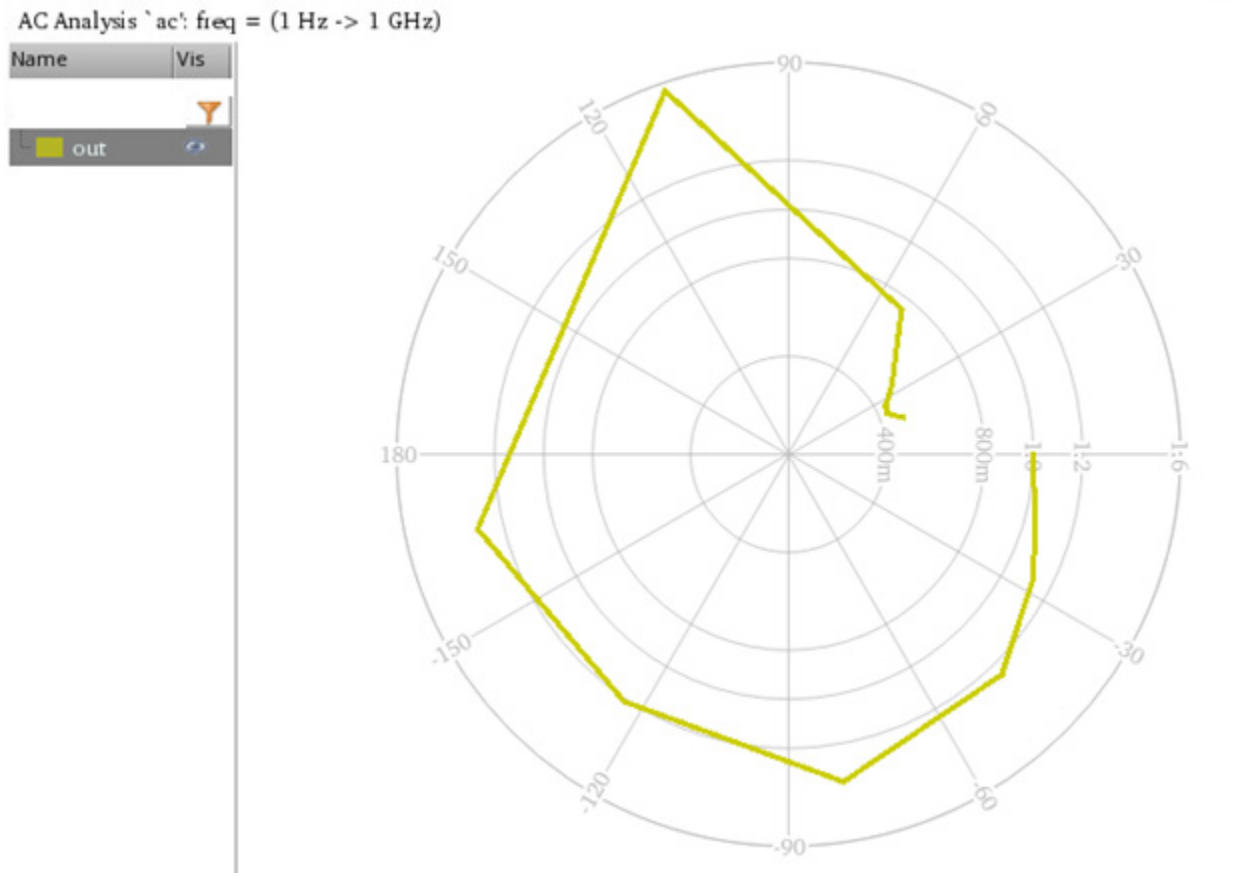
```
awvCreatePlotWindow()
=> window:3
```

The following example opens the simulation results stored in the specified directory `ampsim.raw`.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

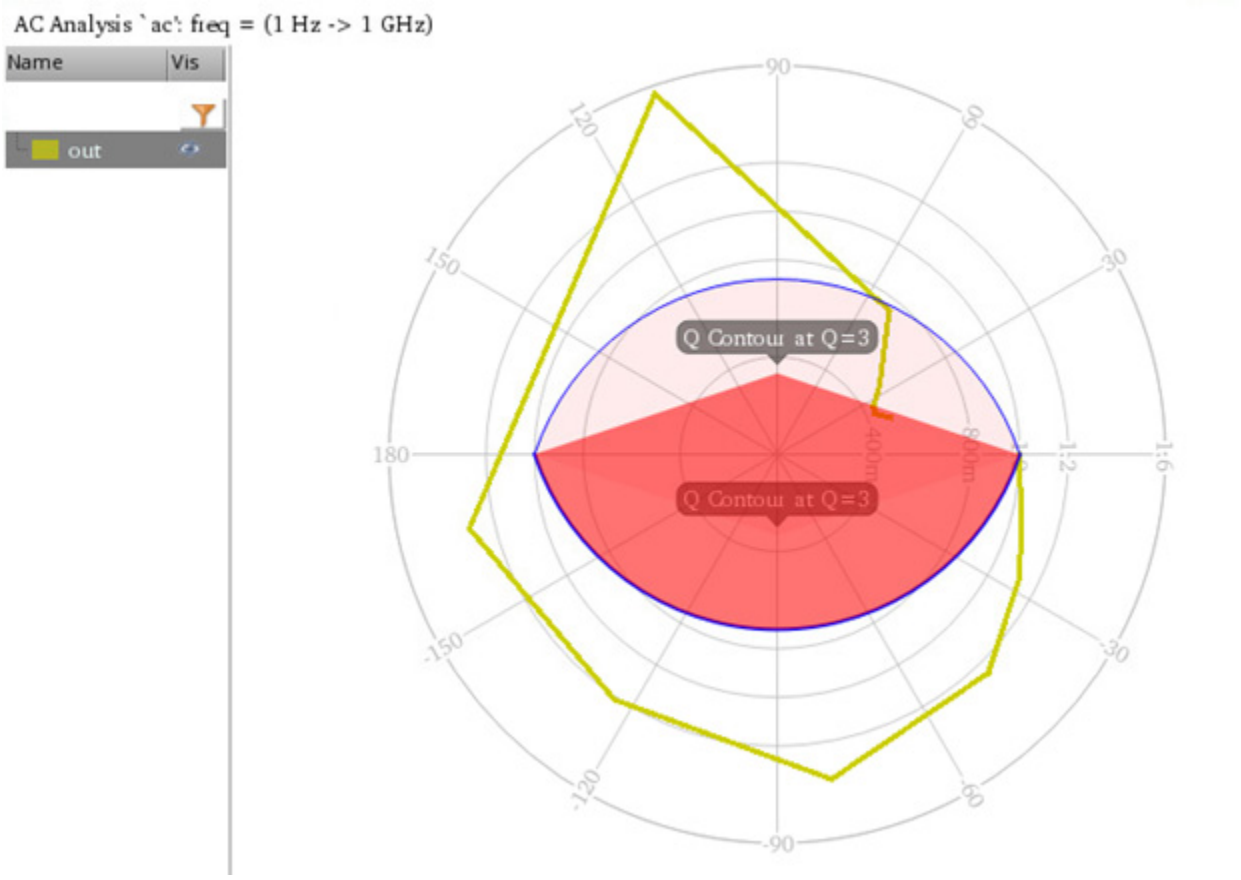
The following example creates plots the signal `out` from the `ac-ac` results of the specified results directory as a `polar` graph in the current Waveform window.

```
awvPlotSignals('(("/servers/user/design/ampsim.raw" (("ac-ac" ("out")))))
?plotStyle "Append" ?graphType "Polar")
=> t
```



The following example adds two Q contour curves for the Q value `3` to the circular graph in the current Waveform window.

```
awvPlaceQContour(awvGetCurrentWindow() 3 ?label "Q Contour at Q=3" ?outlineColor
"blue" ?fillColor "red")
=> ("circleMarker[1.1.1]" "circleMarker[1.1.2]")
```



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

Adds a graph label or a marker label to the specified trace in a subwindow.

Arguments

w_windowID	Waveform window ID.
x_waveIndex	Integer identifying the waveform. Use the `awvGetWaveNameList` function to return the number and names of the waveforms.
l_location	A list of coordinates that describe the location of the label.
t_label	Text to display on the label.
t_expr	String containing an expression that is evaluated when the command is run, and re-evaluated at the completion of each simulation in auto-update mode. You can also set this argument to `nil`.

`?textOffset` *g_textOffset*

Specifies whether to place a graph label or a marker label.

Valid values are:

- `t`: Places a point marker and the corresponding marker label.
- `nil`: Places a graph label.

The default value is `t`.

`?color` *`t_color`*
Color of the label. Available colors are defined in your technology file.

Valid values are from `y1` through `y66`.

`?justify` *`t_justify`*

Justification for the label text.

Valid values are `lowerLeft`, `centerLeft`, `upperLeft`, `lowerCenter`, `centerCenter`, `upperCenter`, `lowerRight`, `centerRight`, and `upperRight`.

`?fontStyle` *`t_fontStyle`*

Font style for the label text.

Valid Values are `stick`, `fixed`, `euroStyle`, `gothic`, `math`, `roman`, `script`, `swedish`, and `milSpec`.

`?height` *`t_height`*
Height of the label.

Valid values are `small`, `medium`, and `large`.

`?orient` *`t_orient`*
Orientation of the label.

Valid values are:

- `R0`: Sets the orientation to horizontal.
- `R90`: Sets the orientation to vertical.

`?subwindow` *`x_subwindow`*

Number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

s_labelID	Identification number of the graph label or the marker label.
nil	The graph label or the marker label cannot be placed because of an error.

Examples

The following example returns the trace numbers and waveform names plotted in subwindow 2 of the current Waveform window.

```
awvGetWaveNameList(awvGetCurrentWindow() ?subwindow 2)
=>
((2 5)
    ("out" "net10")
)
```

The trace `out` is identified with the index number `2` and `net10` is identified with the trace number `5`.

The following example places a point marker and a label `markerLabelonTraceNet10` on the trace `net10` at `x=50ns` and `y=0.5` in subwindow `2` of the current Waveform window.

```
awvPlaceWaveformLabel(awvGetCurrentWindow() 5 list(50ns 0.5)
"markerLabelonTraceNet10" nil ?textOffset t ?subwindow 2)
=> ("pointMarker[45.79.17]")
```

The following example places a point marker and a label `markerLabelonTraceOut` on the trace `out` at `x=50ns` and `y=2.5` in subwindow `2` of the current Waveform window.

```
awvPlaceWaveformLabel(awvGetCurrentWindow() 2 list(50ns 2.5)
"markerLabelonTraceOut" nil ?textOffset t subwindow 2)
=> ("pointMarker[45.79.18]")
```

The following example places a graph label `graphLabelonTraceOut` on the trace `out` at `x=0` and `y=1` in subwindow `2` of the current Waveform window.

```
awvPlaceWaveformLabel(awvGetCurrentWindow() 2 list(0 1) "graphLabelonTraceOut" nil
?textOffset nil ?subwindow 2)
=> ("graphLabel[45.79.24]")
```

The following example places a graph label `graphLabelonTraceNet10` on the trace `net10` at `x=1` and `y=1` in subwindow `2` of the current Waveform window.

```
awvPlaceWaveformLabel(awvGetCurrentWindow() 5 list(1 1) "graphLabelonTraceNet10"
nil ?textOffset nil ?subwindow 2)
=> ("graphLabel[45.79.25]")
```

Note the following points:

- The valid location of a graph label ranges between absolute x-axis and y-axis coordinates (`0`,`0`) and (`1`,`1`).
- The valid location of a marker ranges between data coordinates defined by x axis and y axis limits.

The following example returns `nil` and reports an error because the specified location for the graph label is not within the range (`0`,`0`) and (`1`,`1`).

```
awvPlaceWaveformLabel(awvGetCurrentWindow() 2 list(0 2) "graphLabelonTraceOut" nil
?textOffset nil ?subwindow 2)
The location specified for placing a label on the graph is invalid. Specify a valid
label location between the following ranges (upper and lower bounds inclusive).
x-axis: (0,1)
y-axis: (0,1)
=> nil
```

The following example returns `nil` and reports an error because the specified location for the marker label is outside of the x-axis and y-axis limits of the graph:

```
awvPlaceWaveformLabel(awvGetCurrentWindow() 5 list(600ns 0.5)
"markerLabelonTraceNet10" nil ?textOffset t ?subwindow 2)
The location specified for placing a marker on the graph is invalid. Specify a valid
marker location between the following ranges (upper and lower bounds inclusive).
x-axis: (0.0,5e-07)
y-axis: (-3.388872,4.206791)
=> nil
```

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

Adds a graph label to a subwindow in the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
l_location	A list of coordinates that describe the location of the label.
	The valid location of a graph label ranges between absolute x-axis and y-axis coordinates (0,0) and (1,1).
t_label	Text to display on the label.
t_expr	String containing an expression that is evaluated when the command is run, and re-evaluated at the completion of each simulation in auto-update mode.
	You can also set this argument to nil.
?color *t_color*	Color of the label. Available colors are defined in your technology file.
	Valid values are from y1 through y66.
?justify *t_justify*	
	Justification for the label text.
	Valid values are lowerLeft, centerLeft, upperLeft, lowerCenter, centerCenter, upperCenter, lowerRight, centerRight, and upperRight.

`?fontStyle` *`t_fontStyle`*

Font style for the label text.

Valid Values are `stick`, `fixed`, `euroStyle`, `gothic`, `math`, `roman`, `script`, `swedish`, and `milSpec`.

`?height` *`t_height`* Height of the label.

Valid values are `small`, `medium`, and `large`.

`?orient` *`t_orient`* Orientation of the label.

Valid values are:

- `R0`: Sets the orientation to horizontal.
- `R90`: Sets the orientation to vertical.

`?subwindow` *`x_subwindow`*

Number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

`s_labelID`	Identification number of the graph label.
`nil`	The graph label cannot be placed because of an error.

Examples

The following example places a graph label `windowLabel1` in subwindow `2` of the specified Waveform window.

```
awvPlaceWindowLabel(window(3) list(0 0) "windowLabel1" nil ?subwindow 2)
=> ("graphLabel[1.1.3]")
```

The following example places a graph label `windowLabel2` in subwindow `2` of the specified Waveform window.

```
awvPlaceWindowLabel(window(3) list(0 1) "windowLabel2" nil ?subwindow 2)
=> ("graphLabel[1.1.4]")
```

The following example places a graph label `windowLabel3` in subwindow `2` of the specified Waveform window.

```
awvPlaceWindowLabel(window(3) list(1 1) "windowLabel3" nil ?subwindow 2)
=> ("graphLabel[1.1.5]")
```

The following example places a graph label `windowLabel4` in subwindow `2` of the specified Waveform window.

```
awvPlaceWindowLabel(window(3) list(1 0) "windowLabel4" ?subwindow 2)
=> ("graphLabel[1.1.6]")
```

The following example places a graph label `windowLabel5` in subwindow `2` of the specified Waveform window.

```
awvPlaceWindowLabel(window(3) list(0.5 0.5) "windowLabel5" ?subwindow 2)
=> ("graphLabel[1.1.7]")
```

The following example returns `nil` and reports an error because the specified location of the graph label is not with in the range (`0`,`0`) and (`1`,`1`).

```
awvPlaceWindowLabel(window(3) list(1 2) "windowLabel4" nil ?subwindow 2)
=> nil
```

awvPlaceXMarker

```
awvPlaceXMarker(
     w_windowID
     n_xLoc
     [ ?label t_label ]
     [ ?subwindow x_subwindow ]
     )
     => t_markerID / nil
```

Description

Places a vertical marker at the specified x coordinate in the specified graph window.

Arguments

w_windowID	Waveform window ID
n_xLoc	The x coordinate at which the vertical marker is to be placed.
?label *t_label*	Label to be set on the vertical marker.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

t_markerID	ID of the vertical marker.
nil	The vertical marker cannot be placed on the trace because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

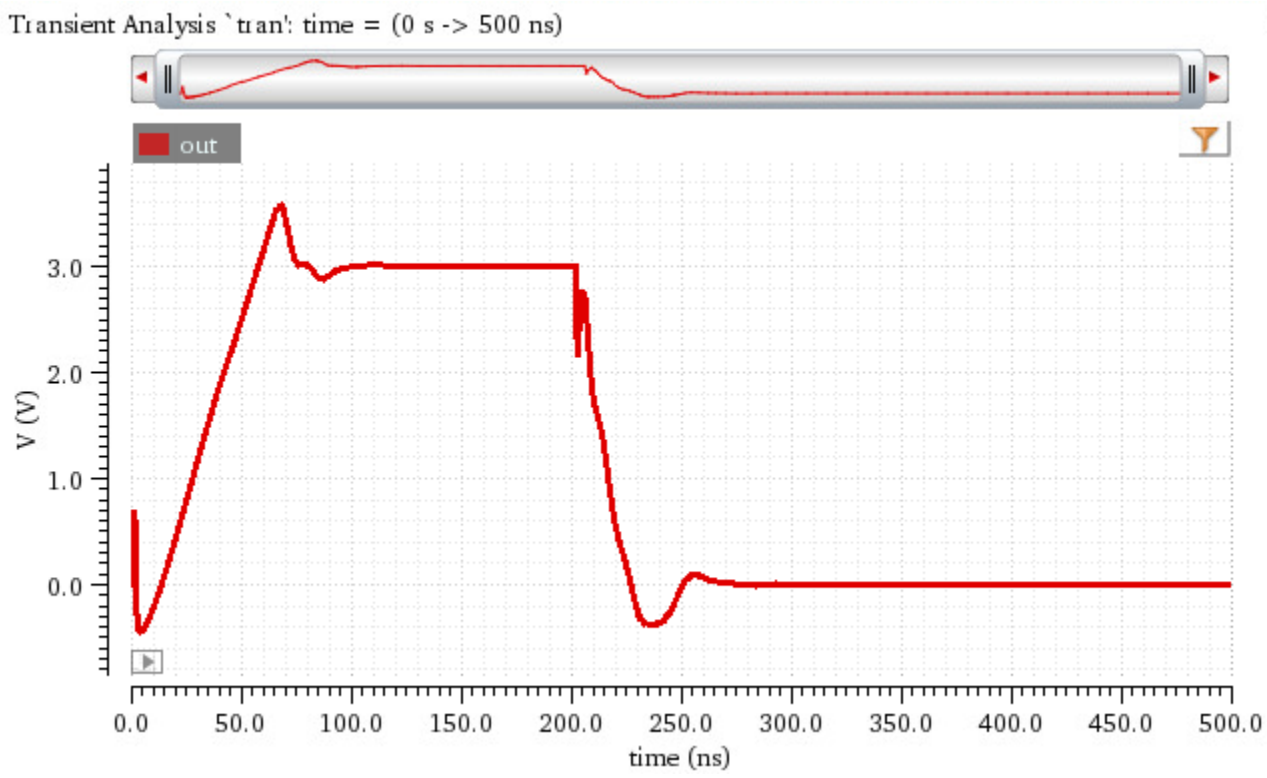
```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

The following example plots signal `out` from the `tran-tran` result of the results directory `ampsim.raw`.

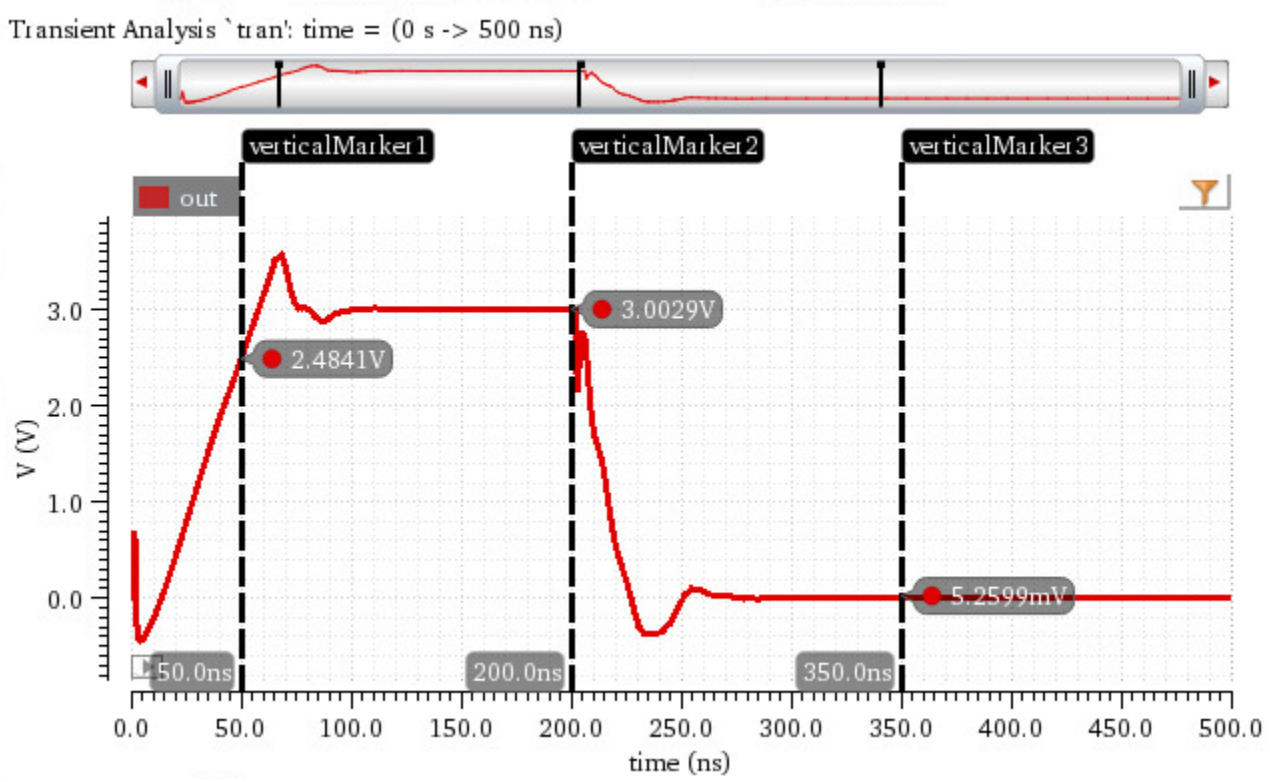
```
plot(v("out" ?result "tran-tran") ?expr "out")
=> t
```



The following examples apply vertical markers on trace `out` at x=`50ns`, x=`200ns`, and x=`350ns` with labels `verticalMarker1`, `verticalMarker2`, and `verticalMarker3` in subwindow `1` of the Waveform window you created using `awvCreatePlotWindow`.

```
awvPlaceXMarker(window(3) 50ns ?label "verticalMarker1" ?subwindow 1)
=> "vertMarker[1.1.1]"
awvPlaceXMarker(window(3) 200ns ?label "verticalMarker2" ?subwindow 1)
=> "vertMarker[1.1.2]"
awvPlaceXMarker(window(3) 350ns ?label "verticalMarker3" ?subwindow 1)
```

```
=> "vertMarker[1.1.3]"
```



awvPlaceYMarker

```
awvPlaceYMarker(
     w_windowID
     n_yLoc
     [ ?label t_label ]
     [ ?subwindow x_subwindow ]
     [ ?stripNum x_stripNumber ]
     )
     => t_markerID / nil
```

Description

Places a horizontal marker at the specified y coordinate on the specified trace.

Arguments

w_windowID	Waveform window ID
n_yLoc	The y coordinate at which the horizontal marker is to be placed.
?label *t_label*	Label to be set on the horizontal marker.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.
?stripNum *x_stripNumber*	Strip number of the trace. If you do not specify this argument, the horizontal marker is placed on the currently selected strip. If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the horizontal marker is placed on the trace with strip number `1`.

Value Returned

t_markerID	ID of the horizontal marker.
nil	The horizontal marker cannot be placed on the trace because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

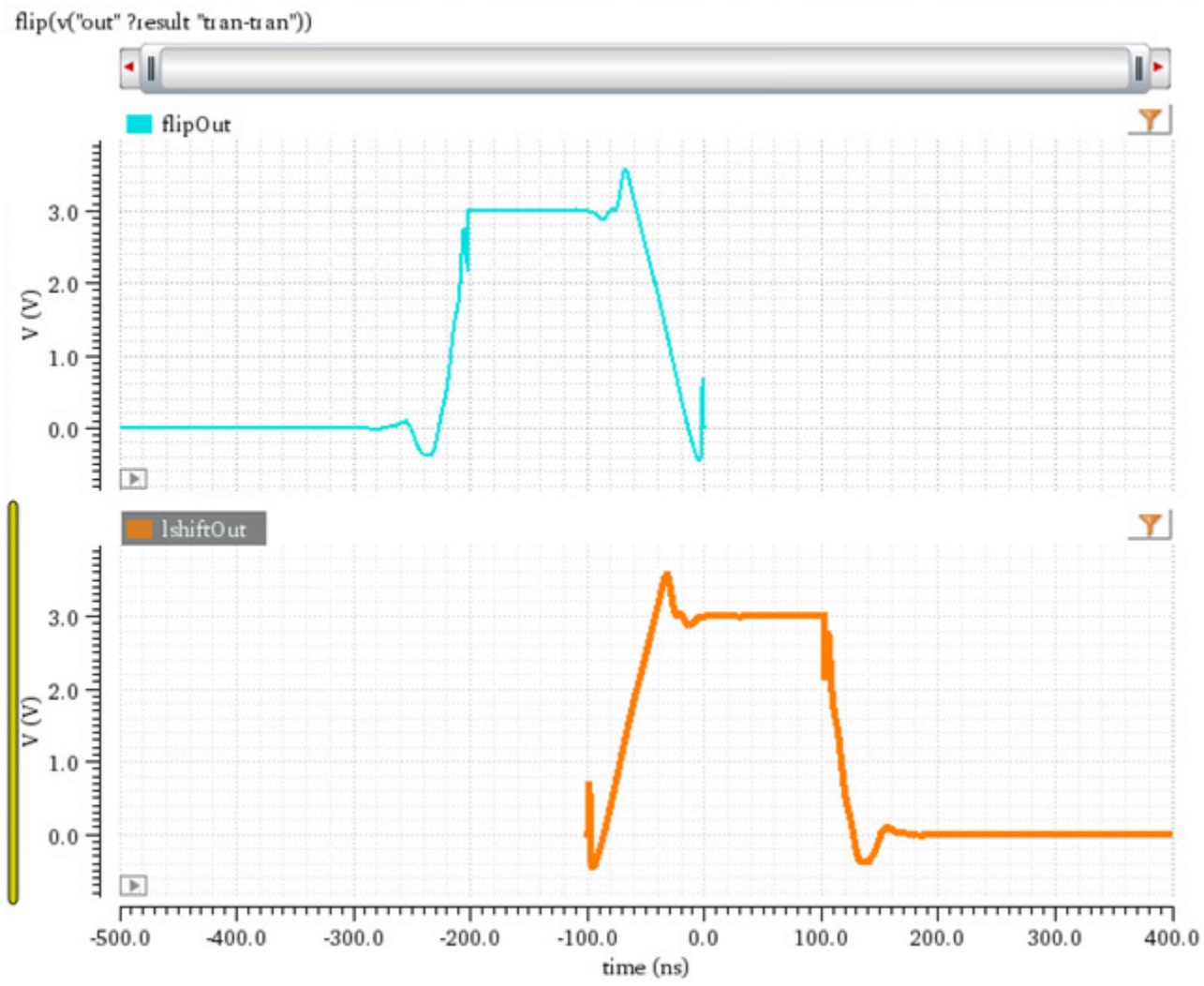
The following example evaluates the expression `flip` applied on the signal `out` and plots the resulting waveform in strip number `1` of the specified Waveform window. The signal `out` is available in the `tran-tran` result of the results directory `ampsim.raw`.

```
awvPlotExpression(window(3) "flip(v(\"out\" ?result \"tran-tran\")" nil ?expr
list("flipOut") ?index list(1) ?color list("y12") ?lineType list("line")
?lineStyle list("solid") ?lineThickness list("medium") ?stripNumber list(1))
=> t
```

The following example evaluates the expression `lshift` applied on the signal `out` and plots the resulting waveform in strip number `2` of the specified Waveform window. The signal `out` is available in the `tran-tran` result of the results directory `ampsim.raw`.

```
awvPlotExpression(window(3) "lshift(v(\"out\" ?result \"tran-tran\") 100ns)" nil
?expr list("lshiftOut") ?index list(2) ?color list("y6") ?lineType list("line")
?lineStyle list("solid") ?lineThickness list("medium") ?stripNumber list(2))
```

```
=> t
```



The following examples apply horizontal markers with label `horizontalMarker1` and `horizontalMarker2` at y=`1.0V` and y=`3.0V` on the trace in strip number `1`.

```
awvPlaceYMarker(window(3) 1.0 ?label "horizontalMarker1" ?stripNum 1)
=> "horizMarker[1.1.1]"
awvPlaceYMarker(window(3) 3.0 ?label "horizontalMarker2" ?stripNum 1)
=> "horizMarker[1.1.2]"
```

The following examples apply horizontal markers with label `horizontalMarker3` and `horizontalMarker4` at y=`0.0V` and y=`2.0V` on the trace in strip number `2`.

```
awvPlaceYMarker(window(3) 0.0 ?label "horizontalMarker3" ?stripNum 2)
=> "horizMarker[1.1.3]"
awvPlaceYMarker(window(3) 2.0 ?label "horizontalMarker4" ?stripNum 2)
```

```
=> "horizMarker[1.1.4]"
```

awvPlotExpression

```
awvPlotExpression(
     w_windowID
     t_expr
     l_context
     [ ?expr l_exprList ]
     [ ?index l_waveIndexList ]
     [ ?color l_colorList ]
     [ ?lineType l_lineTypeList ]
     [ ?lineStyle l_lineStyleList ]
     [ ?lineThickness l_lineThicknessList ]
     [ ?showSymbols l_showList ]
     [ ?dataSymbol l_symbolList ]
     [ ?barBase t_barBase ]
     [ ?barWidth t_barWidth ]
     [ ?barShift t_barShift ]
     [ ?subwindow x_subwindow ]
     )
     => 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_exprList*, 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 when the command is issued and reevaluated at the completion of each simulation in auto-update mode.
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_exprList*	List of waveform names to be displayed in the trace legend.

`?index` *`l_waveIndexList`*

List of index numbers identifying the waveforms.

If you do not specify index numbers for the waveforms in *`l_waveIndexList`*, the lowest unused numbers in the subwindow are assigned.

`?color` *`l_colorList`*

List of colors for the waveforms. Available colors are defined in your technology file. Valid values are from `y1` through `y66`.

If you do not specify this argument, default colors are used.

`?lineType` *`l_lineTypeList`*

List specifying the type of line to be used for the waveforms.

Valid values are `line`, `bar`, `scatterPlot`, `poleZero`.

If you do not specify this argument, default value `line` is used.

`?lineStyle` *`l_lineStyleList`*

List specifying the line style to used for the waveforms.

Valid values are `solid`, `dash`, `dot`, `dashDot`, and `dashDotDot`.

If you do not specify this argument, default value `solid` is used.

`?lineThickness` *`l_lineThicknessList`*

List specifying the line thickness of the waveforms.

Valid values are `fine`, `medium`, `thick`, and `extraThick`.

If you do not specify this argument, default value `fine` is used.

`?showSymbols` *`l_showList`*

List of flags that specify whether to show the symbols on the waveforms. Valid values are `t` and `nil`.

The default value is `nil`, which means that symbols are not displayed on the waveforms.

The number of flags in the *`l_showList`* must match the number of symbols in the *`l_symbolList`*.

`?dataSymbol` *`l_symbolList`*

List of symbols to be displayed for data points on the waveforms.

To use a symbol, specify an integer or a single character corresponding to the symbol.

`?barBase` *`t_barBase`*

Base of the bar.

`?barWidth` *`t_barWidth`*

Width of the bar.

This argument takes integer values. The default value is `1`.

`?barShift` *`t_barShift`*

Specifies whether to shift the bar ahead or backwards when the `?lineType` argument is set to `bar`. This argument takes integer values.

Positive integer shifts the bar backwards and negative integer shifts the bar forward.

The default value is `0`, which indicates that bar is not shifted.

`?subwindow` *`x_subwindow`*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

`?yNumber` *`l_yNumberList`*

List of integers identifying the y axis. Valid values are from `1` through `4`.

`?stripNumber` *`l_stripNumberList`*

List of integers identifying the strips.

Value Returned

`t`	Expression *`t_expr`* is evaluated and the resulting waveforms are plotted in the Waveform window.
`nil`	Expression cannot be evaluated because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

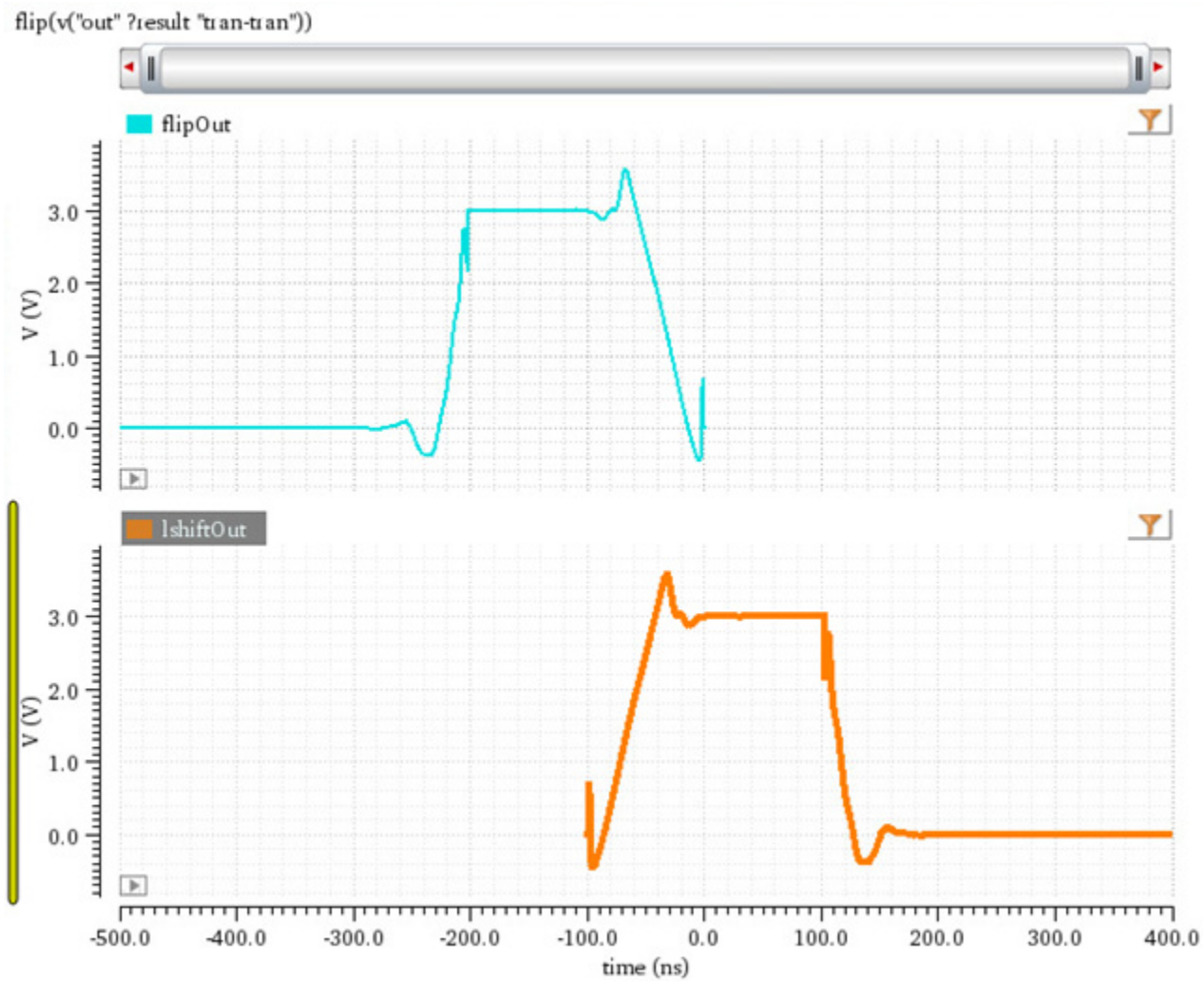
The following example evaluates the expression `flip` applied on the signal `out` and plots the resulting waveform in strip number `1` of the specified Waveform window. The signal `out` is available in the `tran-tran` result of the results directory `ampsim.raw`.

```
awvPlotExpression(window(3) "flip(v(\"out\" ?result \"tran-tran\")" nil ?expr
list("flipOut") ?index list(1) ?color list("y12") ?lineType list("line")
?lineStyle list("solid") ?lineThickness list("medium") ?stripNumber list(1))
=> t
```

The following example evaluates the expression `lshift` applied on the signal `out` and plots the resulting waveform in strip number `2` of the specified Waveform window. The signal `out` is available in the `tran-tran` result of the results directory `ampsim.raw`.

```
awvPlotExpression(window(3) "lshift(v(\"out\" ?result \"tran-tran\") 100ns)" nil
?expr list("lshiftOut") ?index list(2) ?color list("y6") ?lineType list("line")
?lineStyle list("solid") ?lineThickness list("medium") ?stripNumber list(2))
```

```
=> t
```



The following example 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.

```
awvPlotExpression(window(11)
    "expr(x sin(x) linRg(0 1.5 .5))" nil
    ?expr list("sine") ?index list(3) ?color list("y2"))
=> t
```

Related Topics

Symbols for Data Points

awvPlotList

```
awvPlotList(
     w_windowID
     l_yListList
     l_xList
     [ ?expr l_exprList ]
     [ ?index l_waveIndexList ]
     [ ?color l_colorList ]
     [ ?lineType l_lineTypeList ]
     [ ?lineStyle l_lineStyleList ]
     [ ?lineThickness l_lineThicknessList ]
     [ ?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 specified in *l_yListList* against the x-axis values specified in *l_xList* and displays the resulting waveforms in a subwindow.

Arguments

w_windowID	Waveform window ID.
l_yListList	A list of lists that specifies y-axis values. Each list must have the same number of items.
l_xList	A list that specifies x-axis values. Number of items in this list must be equal to the number of items in each list specified by *l_yListList*.
?expr *l_exprList*	List of waveform names to be displayed in the trace legend.
?index *l_waveIndexList*	

List of index numbers identifying the waveforms.

If you do not specify index numbers for the waveforms in *l_waveIndexList*, the lowest unused numbers in the subwindow are assigned.

`?color` *l_colorList*

List of colors for the waveforms. Available colors are defined in your technology file. Valid values are from `y1` through `y66`.

If you do not specify this argument, default colors are used.

`?lineType` *l_lineTypeList*

List specifying the type of line to be used for the waveforms.

Valid values are `line`, `bar`, `scatterPlot`, `poleZero`.

If you do not specify this argument, default value `line` is used.

`?lineStyle` *l_lineStyleList*

List specifying the line style to used for the waveforms.

Valid values are `solid`, `dash`, `dot`, `dashDot`, and `dashDotDot`.

If you do not specify this argument, default value `solid` is used.

`?lineThickness` *l_lineThicknessList*

List specifying the line thickness of the waveforms.

Valid values are `fine`, `medium`, `thick`, and `extraThick`.

If you do not specify this argument, default value `fine` is used.

`?showSymbols` *l_showList*

List of flags that specify whether to show the symbols on the waveforms. Valid values are `t` and `nil`.

The default value is `nil`, which means that symbols are not displayed on the waveforms.

The number of flags in the *l_showList* must match the number of symbols in the *l_symbolList*.

`?dataSymbol` *l_symbolList*

List of symbols to be displayed for data points on the waveforms.

To use a symbol, specify an integer or a single character corresponding to the symbol.

`?barBase` *`t_barBase`*

Base of the bar.

`?barWidth` *`t_barWidth`*

Width of the bar.

This argument takes integer values. The default value is `1`.

`?barShift` *`t_barShift`*

Specifies whether to shift the bar ahead or backwards when the `?lineType` argument is set to `bar`. This argument takes integer values.

Positive integer shifts the bar backwards and negative integer shifts the bar forward.

The default value is `0`, which indicates that bar is not shifted.

`?subwindow` *`x_subwindow`*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

`?yNumber` *`l_yNumberList`*

List of integers identifying the y axis. Valid values are from `1` through `4`.

`?stripNumber` *`l_stripNumberList`*

List of integers identifying the strips.

Value Returned

`t`	Specified y-axis values are plotted against the specified x-axis values and the corresponding waveforms are displayed in the specified window.
`nil`	Waveforms cannot be plotted because of an error.

Examples

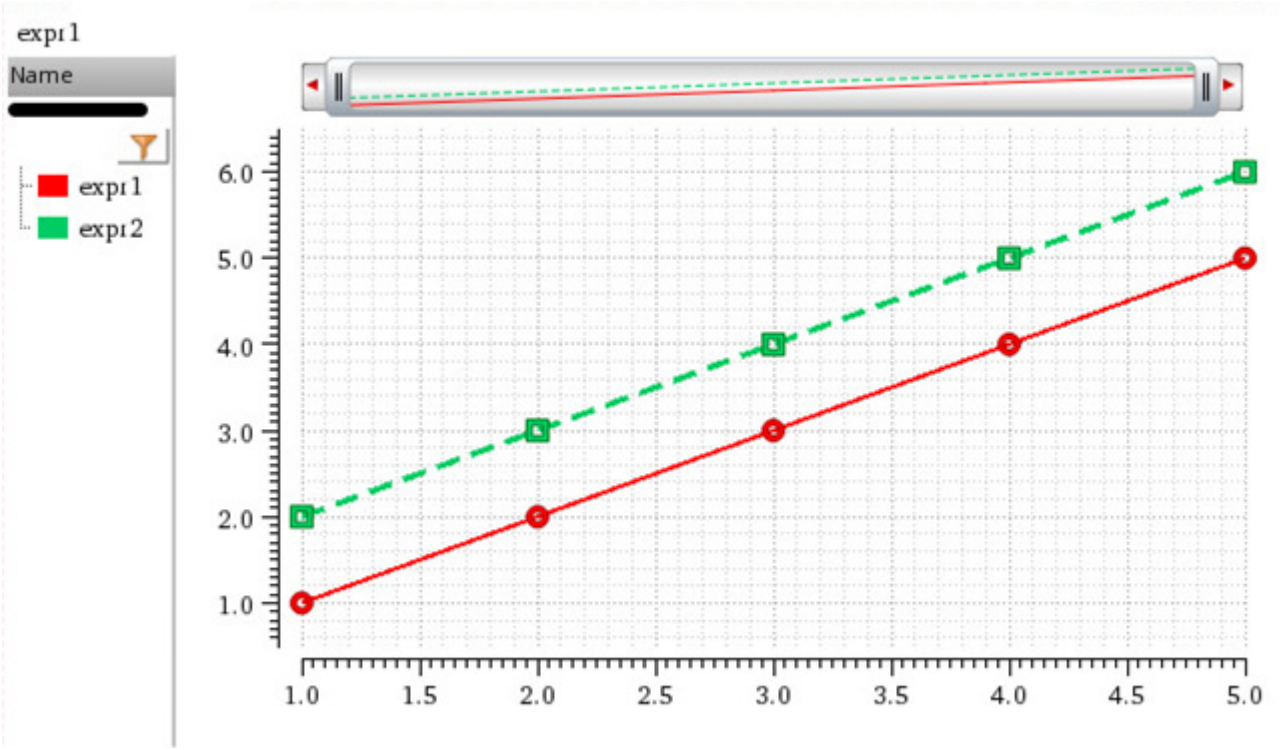
The following example creates a Waveform window and returns the ID of the window.

```
awvCreatePlotWindow()
=> window:3
```

The following example plots the y-axis values (`1`, `2`, `3`) and (`2`, `3`, `4`) against the x-axis values (`1`, `2`, `3`) as waveforms with names `expr1` and `expr2`. The waveforms `expr1` and `expr2` are identified with the index numbers `1` and `2`, respectively.

```
awvPlotList(window(3) list(list(1 2 3) list(2 3 4)) list(1 2 3) ?expr list("expr1"
"expr2") ?index list(1 2) ?color list("y1" "y66") ?lineType list("line" "line")
?lineStyle list("dashDotDot" "solid") ?lineThickness list("extraThick" "fine")
?showSymbols list(t t) ?dataSymbol list(4 5))
=> t
```

Note that `expr1` uses a symbol corresponding to integer `4`, which is a `circle`. `expr2` uses a symbol corresponding to integer `5`, which is a `square`.



awvPlotSignals

```
awvPlotSignals(
     l_signalList
     [ ?plotStyle t_plotStyle ]
     [ ?graphType t_graphType ]
     [ ?graphModifier t_graphModifier ]
     [ ?waveType g_waveType ]
     )
     => t / nil
```

Description

Plots the signals specified in a list.

Arguments

l_signalList List of signals to be plotted.

Specifies the list of signals in the following format:

`(list (list` *resultsDir1* `(list (list` *result1* `(list` *signal1 signal2* `...) ...) ...)))`

`?plotStyle` *t_plotStyle*

Plotting style.

Valid values are `Append`, `Replace`, `New Window`, and `New Subwindow`.

`?graphType` *t_graphType*

Type of graph.

Valid values are `Default`, `Rectangular`, `Polar`, `Impedance`, `Admittance`, `Immittance`, and `RealvsImag`.

`?graphModifier` *t_graphModifier*

Y axis of rectangular graphs.

Valid values are `Magnitude`, `Phase`, `WPhase`, `Real`, `Imaginary`, `dB10`, `dB20`.

`?waveType` *g_waveType*

Specifies whether the signal is Y versus Y or not.

Valid values are `YvsY` and `nil`.

Value Returned

`t`	Signals are plotted successfully.
`nil`	Signals cannot be plotted because of an error.

Examples

The following example plots the signals `net10` and `out` from the results `tran-tran` in the results directory `ampsim.raw`.

```
awvPlotSignals('(("/servers/user/design/ampsim.raw" (("tran-tran" ("net10"
"out"))))))
```

The following example plots the signals `net10` and `out` from the results `tran-tran` in the results directory `ampsim.raw`, and signals `jitter` and `delay` from results `tran-tran` in the results directory `prbs.raw`.

```
awvPlotSignals('(("/servers/user/design/ampsim.raw" (("tran-tran" ("net10"
"out")))) ("/servers/user/design/prbs.raw" (("tran-tran" ("jitter" "delay"))))))
```

The following example plots the signals `net10`, `out`, `jitter`, and `delay`. Note that `net10` and `out` signals have `dB10` as their y-axis because these signals have been plotted from an `ac` results.

```
awvPlotSignals('(("/servers/user/design/ampsim.raw" (("ac-ac" ("net10" "out"))))
("/servers/user/design/prbs.raw" (("tran-tran" ("jitter" "delay")))))
?graphModifier "dB10")
```

awvPlotSimpleExpression

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

Description

Evaluates the specified expression and plots the resulting waveform.

Arguments

t_expression Expression to be evaluated and plotted.

?plotStyle *t_plotStyle*

Plotting style.

Valid values are `Append`, `Replace`, `New Window`, and `New Subwindow`.

?graphType *t_graphType*

Type of graph.

Valid values are `Default`, `Rectangular`, `Polar`, `Impedance`, `Admittance`, `Immittance`, and `RealvsImag`.

?graphModifier *t_graphModifier*

Y axis of rectangular graphs.

Valid values are `Magnitude`, `Phase`, `WPhase`, `Real`, `Imaginary`, `dB10`, `dB20`.

?waveType *g_waveType*

Specifies whether the signal is Y versus Y or not.

Valid values are `YvsY` and `nil`.

Value Returned

`t`	Expression is evaluated and the resulting waveform is plotted successfully.
`nil`	Expression cannot be evaluated because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens the results stored in the directory `ampsim.raw`. The `tran-tran` result in this directory contains the signals `net35` and `net10` that are used in the expression to be evaluated and plotted.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

The following example evaluates the specified expression and plots the resulting waveform in subwindow `1` of the Waveform window that you created using `awvCreatePlotWindow`.

```
awvPlotSimpleExpression("v(\"net35\" ?result \"tran-tran\")-v(\"net10\" ?result
\"tran-tran\")" ?plotStyle "Append" ?graphType "" ?graphModifier "Magnitude")
=> t
```

The following example opens the results stored in the directory `prbs.raw`. The `tran-tran` result in this directory contain a signal `jitter` that is used in the expression to be evaluated and plotted.

```
openResults("/servers/user/design/prbs.raw")
=> "/servers/user/design/prbs.raw"
```

The following example evaluates the specified expression and plots the resulting eye diagram waveform in subwindow `2` of the Waveform window that you created using `awvCreatePlotWindow`.

```
awvPlotSimpleExpression("eyeDiagram(v(\"jitter\" ?result \"tran-tran\"
?resultsDir \"./prbs.raw\") 200n 400u 2*40n   ?autoCenter t )" ?plotStyle "New
SubWindow" ?graphModifier "Magnitude")
=> t
```

awvPlotWaveform

```
awvPlotWaveform(
     w_windowID
     l_waveformList
     [ ?subwindow x_subwindow ]
     [ ?expr l_exprList ]
     [ ?index l_waveIndexList ]
     [ ?component t_component ]
     [ ?color l_colorList ]
     [ ?lineType l_lineTypeList ]
     [ ?lineStyle l_lineStyleList ]
     [ ?lineThickness l_lineThicknessList ]
     [ ?showSymbols l_showList ]
     [ ?dataSymbol l_symbolList ]
     [ ?barBase t_barBase ]
     [ ?barWidth t_barWidth ]
     [ ?barShift t_barShift ]
     [ ?yaxisUnit t_yaxisUnit ]
     [ ?yaxisLabel t_yaxisLabel ]
     [ ?modifierType t_modifierType ]
     [ ?graphType t_graphType ]
     [ ?stripNumber x_stripNumber ]
     )
     => t / nil
```

Description

Plots the waveforms in the list *l_waveformList*.

Arguments

w_windowID Waveform window ID.

l_waveformList A list of waveform objects to be plotted.

?subwindow *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, waveforms are plotted in the current subwindow of the specified Waveform window.

?expr *l_exprList*

List of waveform names to be displayed in the trace legend.

?index *l_waveIndexList*

List of index numbers identifying the waveforms.

If you do not specify index numbers for the waveforms in *l_waveIndexList*, the lowest unused numbers in the subwindow are assigned.

?component *t_component*

Plot option for the waveform. This option is ignored unless display mode is set to `strip` or `composite`.

The imaginary part of the waveform is plotted against its real part when display mode is set to `smith`.

Valid values are:

- `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.

Default value is `real`.

?color *l_colorList*

List of colors for the waveforms. Available colors are defined in your technology file. Valid values are from `y1` through `y66`.

If you do not specify this argument, default colors are used.

The color information can also be provided in the RGB format, such as `0X00FF50`.

`?lineType` *`l_lineTypeList`*

List specifying the type of line to be used for the waveforms.

Valid values are `line`, `bar`, `scatterPlot`, `poleZero`.

If you do not specify this argument, default value `line` is used.

`?lineStyle` *`l_lineStyleList`*

List specifying the line style to used for the waveforms.

Valid values are `solid`, `dash`, `dot`, `dashDot`, and `dashDotDot`.

If you do not specify this argument, default value `solid` is used.

`?lineThickness` *`l_lineThicknessList`*

List specifying the line thickness of the waveforms.

Valid values are `fine`, `medium`, `thick`, and `extraThick`.

If you do not specify this argument, default value `fine` is used.

`?showSymbols` *`l_showList`*

List of flags that specify whether to show the symbols on the waveforms. Valid values are `t` and `nil`.

The default value is `nil`, which means that symbols are not displayed on the waveforms.

The number of flags in the *`l_showList`* must match the number of symbols in the *`l_symbolList`*.

`?dataSymbol` *`l_symbolList`*

List of symbols to be displayed for data points on the waveforms.

To use a symbol, specify an integer or a single character corresponding to the symbol.

`?barBase` *`t_barBase`*

Specifies base of the bar when the `?lineType` argument is set to `bar`.

`?barWidth` *`t_barWidth`*

Specifies width of the bar when the `?lineType` argument is set to `bar`.

This argument takes integer values. The default value is `1`.

`?barShift` *`t_barShift`*

Specifies whether to shift the bar ahead or backwards when the `?lineType` argument is set to `bar`. This argument takes integer values.

Positive integer shifts the bar backwards and negative integer shifts the bar forward.

The default value is `0`, which indicates that bar is not shifted.

`?yaxisUnit` *`l_yaxisUnit`*

List of units for y axis.

Default value is an empty string.

`?yaxisLabel` *`l_yaxisLabel`*

List of labels of y axis.

Default value is an empty string.

`?modifierType` *`t_modifierType`*

List of graph modifiers.

`?graphType` *`t_graphType`*

List of graph types.

Valid Values are `rectangular`, `polar`, `impedance`, `admittance`, and `immittance`.

Default value is `rectangular`.

`?stripNumber` *`x_stripNumber`*

A list of integers identifying the strip numbers of the traces to be plotted. If you do not specify this argument, the currently selected strip is used to plot traces.

Value Returned

`t`	Waveforms specified in the *`l_waveformList`* list are plotted in the specified Waveform window.
`nil`	Waveforms cannot be plotted because of an error.

Examples

The following example creates a Waveform window and returns its ID.

```
awvCreatePlotWindow()
=> window:3
```

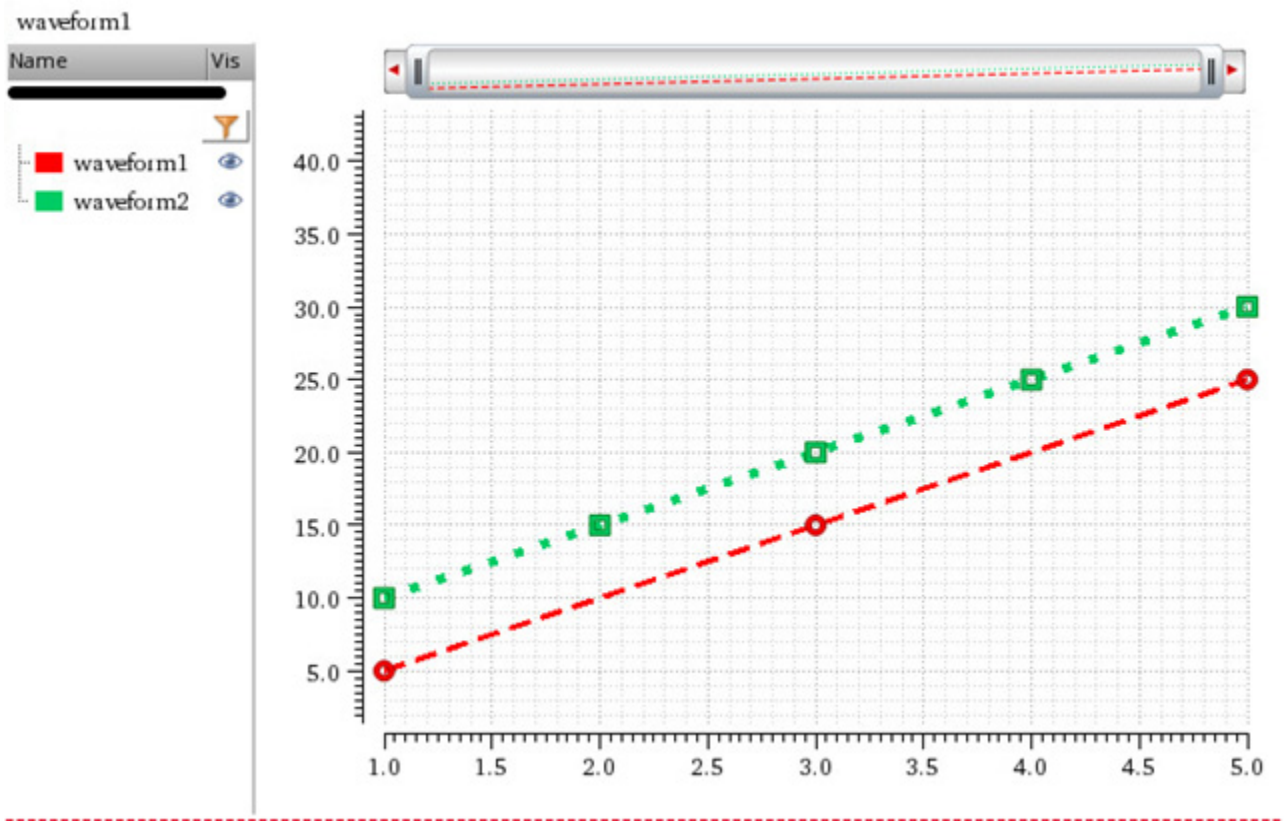
The following examples create waveform objects `w1`, `w2`, `w3`, and `w4`.

```
w1=drCreateWaveform(drCreateVec('double list(1 3 5)) drCreateVec('double list(5 15
25)))
=> srrWave:0x3208c020
w2=drCreateWaveform(drCreateVec('double list(1 2 3 4 5)) drCreateVec('double
list(10 15 20 25 30)))
=> srrWave:0x3208c030
w3=drCreateWaveform(drCreateVec('double list(1 3 5)) drCreateVec('double list(15
25 35)))
=> srrWave:0x3208c040
w4=drCreateWaveform(drCreateVec('double list(1 2 3 4 5)) drCreateVec('double
list(20 25 30 35 40)))
=> srrWave:0x3208c050
```

The following example plots the waveforms objects `w1` and `w2` specified in the list.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(w1 w2)
                ?expr list("waveform1" "waveform2")
                ?color list("y1" "y66")
                ?index list(1 2)
                ?lineType list("line" "line")
                ?lineStyle list("dash" "dot")
                ?lineThickness list("thick" "extraThick")
                ?showSymbols list(t t)
                ?dataSymbol list(4 5)
                )
```

```
=> t
```



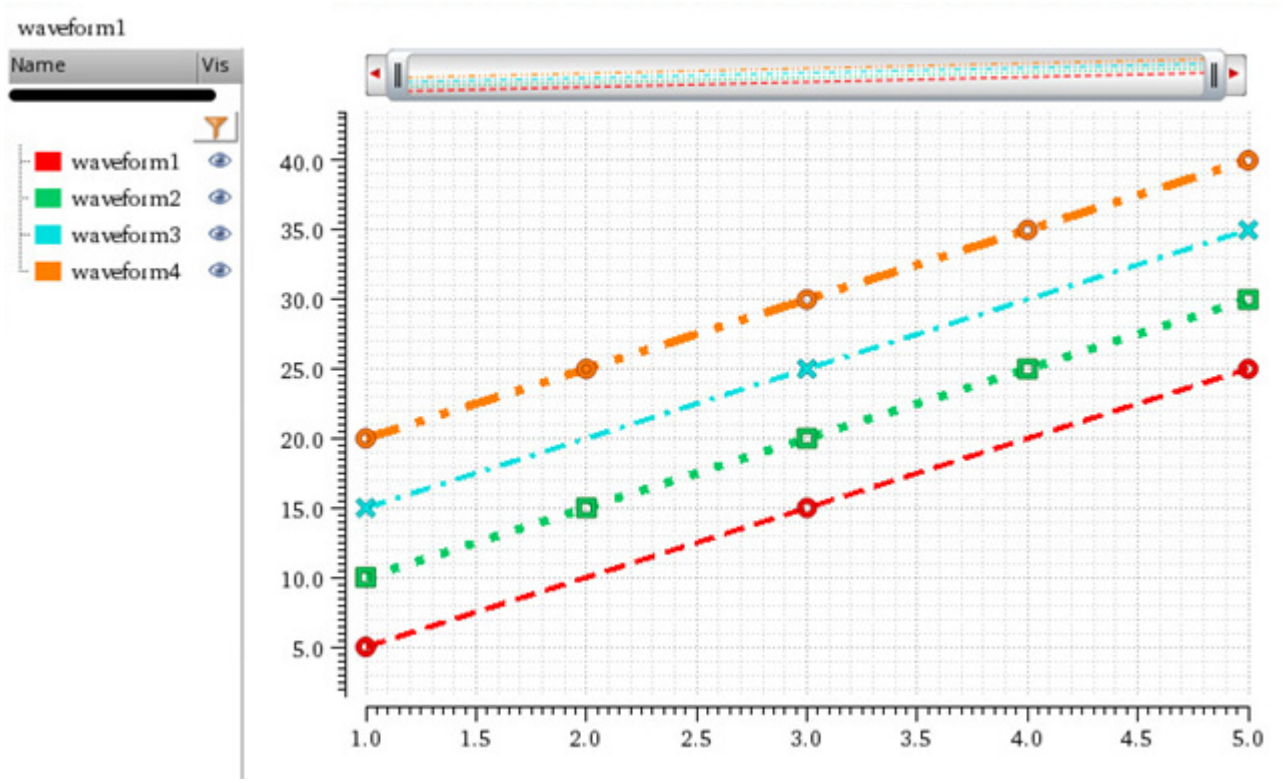
The following example returns the index numbers and names of the waveforms plotted using `awvPlotWaveform` function.

```
awvGetWaveNameList(window(3))
=>
((1 2)
    ("waveform1" "waveform2")
)
```

The following example plots the waveform objects `w3` and `w4` specified in the list and appends them to the waveforms plotted using the `awvPlotWaveform` function.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(w3 w4)
                ?expr list("waveform3" "waveform4")
                ?color list("y12" "y6")
                ?index list(5 6)
                ?lineType list("line" "line")
                ?lineStyle list("dashDot" "dashDotDot")
```

```
                ?lineThickness list("thick" "extraThick")
                ?showSymbols list(t t)
                ?dataSymbol list("x" "O")
                )
=> t
```



The following example returns the index numbers and names of the waveforms plotted in the specified Waveform window.

```
awvGetWaveNameList(window(3))
((1 2 5 6)
    ("waveform1" "waveform2" "waveform3" "waveform4")
)
```

The following example creates a Waveform window and returns its ID.

```
win=awvCreatePlotWindow()
=> window:3
```

The following examples create waveform objects `w1`, `w2`, `w3`, and `w4`.

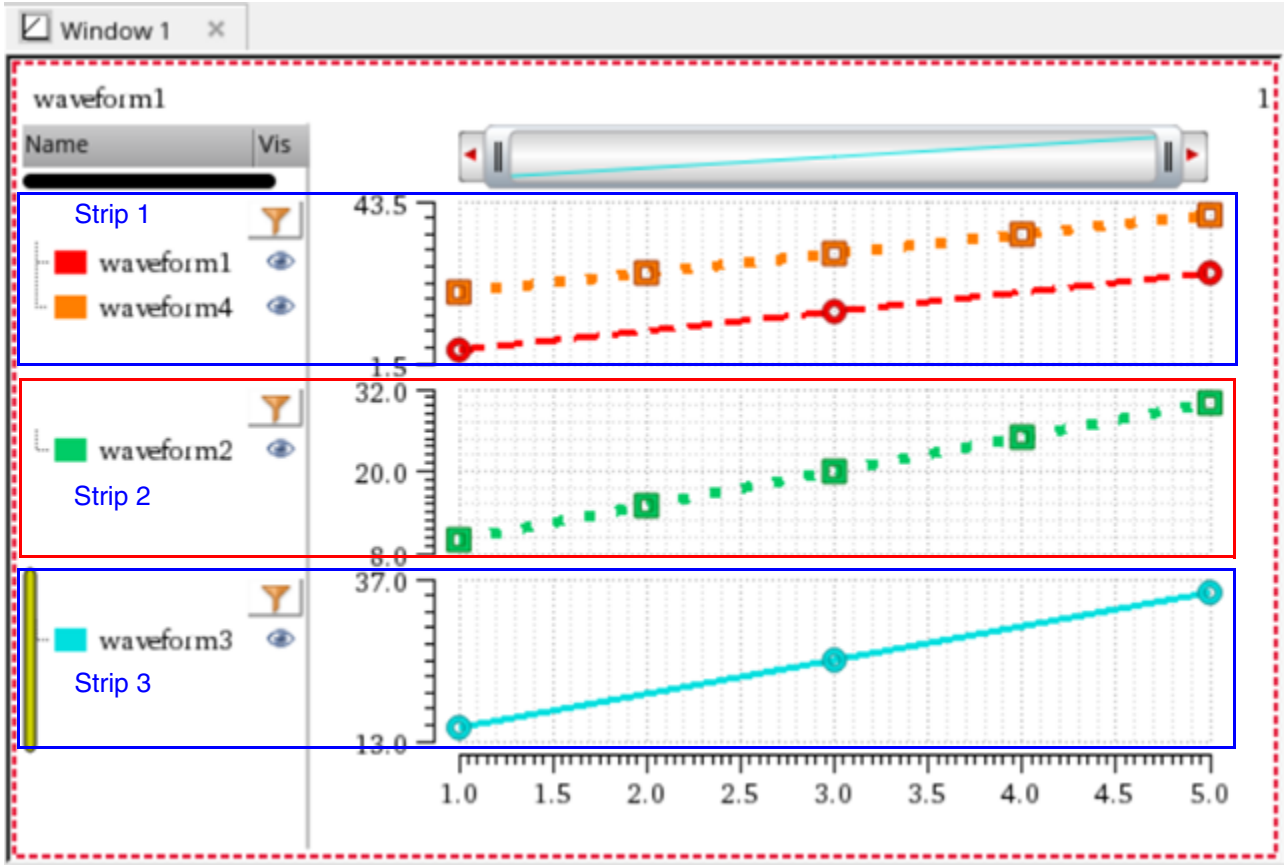
```
w1=drCreateWaveform(drCreateVec('double list(1 3 5)) drCreateVec('double list(5 15
25)))
```

```
=> srrWave:0x31d37020
w2=drCreateWaveform(drCreateVec('double list(1 2 3 4 5)) drCreateVec('double
list(10 15 20 25 30)))
=> srrWave:0x31d37030
w3=drCreateWaveform(drCreateVec('double list(1 3 5)) drCreateVec('double list(15
25 35)))
=> srrWave:0x31d37040
w4=drCreateWaveform(drCreateVec('double list(1 2 3 4 5)) drCreateVec('double
list(20 25 30 35 40)))
=> srrWave:0x31d37050
```

The following example plots the waveforms `w1`, `w2`, `w3`, and `w4` in the specified Waveform window win. Note that the waveforms `w1` and `w4` are plotted in a single strip with strip number `1`. Whereas, the waveforms `w2` and `w3` are plotted in different strips with strip numbers `2` and `3`, respectively.

```
awvPlotWaveform(
                win
                list(w1 w2 w3 w4)
                ?expr list("waveform1" "waveform2" "waveform3" "waveform4")
                ?color list("y1" "y66" "y12" "y6")
                ?index list(1 2 3 4)
                ?lineType list("line" "line" "line" "line")
                ?lineStyle list("dash" "dot" "dash" "dot")
                ?lineThickness list("thick" "extraThick" "thick" "extraThick")
                ?showSymbols list(t t t t)
                ?dataSymbol list(4 5 4 5)
                ?stripNumber list(1 2 3 1)
)
```

```
=> t
```



Returns the strip numbers of traces plotted in the specified Waveform window.

```
awvGetStripNumbersList(win)
=> (1 2 3)
```

Returns the strip number in which the currently selected waveform is plotted.

```
awvGetStripNumberOfSelectedTrace(win)
=> 3
```

Related Topics

Symbols for Data Points

awvPlotWaveformOption

```
awvPlotWaveformOption(
     w_windowID
     x_waveIndex
     t_plotOption
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the specified plotting option for a waveform in a subwindow.

Arguments

w_windowID	Waveform window ID.
x_waveIndex	Index number of the waveform. You can use the `awvGetWaveNameList` function to return a list of index numbers of the waveforms plotted in a subwindow.

t_plotOption Specifies the plotting option of the specified waveform.

Valid values are:

- `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.

`?subwindow` *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

`t` The specified plotting option for the waveform is set successfully.

`nil`	The specified Waveform window, subwindow, or the waveform index does not exist.

Examples

The following example returns the indexes and signal names of the waveforms plotted in the subwindow `2` of the current Waveform window.

```
win=awvGetCurrentWindow()
=> window:4
awvGetWaveNameList(win ?subwindow 2)
=>
((26 17 18 19)
    ("/net027" "/net028" "/net029" "/net030")
)
```

The following example sets the plotting option for the waveform `net029` to `dB20` in the subwindow `2` of the specified Waveform window.

```
awvPlotWaveformOption(
                        window(4)
                        18
                        "dB20"
                        ?subwindow 2
                        )
=> t
```

awvPrintWaveform

```
awvPrintWaveform(
     o_waveform1 [ o_waveform2 ... ]
     [ ?output t_fileName | port ]
     [ ?numSigDigits x_numSigDigits ]
     [ ?format s_format ]
     [ ?numSpaces x_numSpaces ]
     [ ?width x_width ]
     [ ?from x_from ]
     [ ?to x_to ]
     [ ?step x_stepValue ]
     )
     => t / nil
```

Description

Prints text data of waveforms specified in a list of waveforms to the Results Display Window or to the specified file.

There is a limitation of `awvPrintWaveform` function for precision. It works up to `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

o_waveform1 Waveform object.

?output *t_fileName*

Name of the file in which you want to print waveform data.

?numSigDigits *x_numSigDigits*

The number of significant digits to print.

This value overrides any global precision value set with the setup command.

Valid values are integer values from `1` through `16`.

Default value is `6`.

`?format` *`s_format`*	Format of notation in which information is printed. Valid values are: ■ `'engineering`: Uses the engineering notation. For example, `28.88E-9s`. ■ `'scientific`: Uses the scientific notation. For example, `2.888E-8s`. ■ `'suffix`: Uses the suffix notation. For example, `28.88ns`. ■ `'none`: Turns off formatting, which helps you speed up printing large data. To further speed up printing, use the `'none` value and set the `?output` argument to a filename or a port, so that output does not go to CIW. Default value is `'suffix`.
`?numSpaces` *`x_numSpaces`*	The number of spaces between columns. Valid values are integers greater than or equal to `1`. Default value is `4`.
`?width` *`x_width`*	Width of each column. Valid values are integers greater than or equal to `4`. Default value is `14`.
`?from` *`x_from`*	x-axis value after which you want to save waveform data.
`?to` *`x_to`*	x-axis value upto which you want to save waveform data.
`?step` *`x_step`*	Step value to be used.

Value Returned

`t`	Waveform data is printed.
`nil`	Waveform data cannot be printed because of an error.

Examples

The following example creates a Waveform window and returns the ID of the Waveform window.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified results directory `ampsim.raw`.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

The following example returns ID of the waveform of the signal `out`, which is available in the `tran-tran` result of the results directory `ampsim.raw`.

```
w1=v("out" ?result "tran-tran")
=> srrWave:0x36368020
```

The following example returns ID of the waveform that is obtained after applying the `flip` function to the signal `out`, which is available in the `tran-tran` result of the results directory `ampsim.raw`.

```
w2=flip(v("out" ?result "tran-tran"))
=> srrWave:0x36368040
```

The following example passes waveform objects `w1` and `w2` into a list named `waves`.

```
waves=list(w1 w2)
=> (srrWave:0x36368020 srrWave:0x36368040)
```

The following example prints waveform data for `w1` and `w2` in the Results Display Window.

```
awvPrintWaveform(w1 w2 ?from 100ns ?to 200ns ?step 1ns ?numSpaces 4 ?width 50
?format 'suffix ?numSigDigits 10)
=> t
```

The following example prints waveform data for `w1` and `w2` to `myFile.txt` file.

```
awvPrintWaveform(?from 100ns ?to 200ns ?step 1ns ?numSpaces 4 ?width 50 ?format
'suffix ?numSigDigits 10 ?output "myFile.txt" w1 w2)
=> t
```

awvRedisplaySubwindow

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

Description

Refreshes the display of a subwindow in the specified Waveform window.

Arguments

w_windowID Waveform window ID.

?subwindow *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

?readData *g_readData*

Specifies whether to read simulation data.

Valid values are:

- `t`: Reads simulation data, refreshes the subwindow with new data, and updates traces, if required.
- `nil`: Refreshes display of the subwindow.

Value Returned

`t`	Subwindow is refreshed successfully.
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following example refreshes the display of subwindow `2` of the current Waveform window.

```
awvRedisplaySubwindow(awvGetCurrentWindow() ?subwindow 2 ?readData nil)
=> t
```

awvRedisplayWindow

```
awvRedisplayWindow(
     w_windowID
     [ ?readData g_readData ]
     )
     => t / nil
```

Description

Refreshes the display of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?readData *g_readData*	Specifies whether to read simulation data. Valid values are: ■ `t`: Reads simulation data, refreshes the Waveform window with new data, and updates traces, if required. ■ `nil`: Refreshes display of the Waveform window.

Value Returned

`t`	Waveform window is refreshed successfully.
`nil`	The specified Waveform window does not exist.

Examples

The following example refreshes the display of the current Waveform window.

```
awvRedisplaySubwindow(awvGetCurrentWindow() ?readData nil)
=> t
```

awvRedrawWindowMenuCB

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

Description

Redraws all the objects, such as waveforms and markers in a Waveform window. The function is defined in `dfII/etc/context/awv.cxt`.

Arguments

None

Value Returned

`t`	Objects in the Waveform window are redrawn successfully.
`nil`	Objects cannot be redrawn because of an error.

Examples

The following example shows how to run the `awvRedrawWindowMenuCB` function.

```
awvRedrawWindowMenuCB()
=> nil
```

awvRemoveDate

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

Description

Removes the title from the specified Waveform window.

Arguments

w_windowID	Waveform window ID.

Value Returned

t	Title is removed from the specified Waveform window.
nil	The specified Waveform window does not exist.

Examples

The following example removes the title from the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:77
awvRemoveDate(win)
=> t
```

The following example removes the title from the specified Waveform window.

```
awvRemoveDate(window(78))
=> t
```

awvRemoveLabel

```
awvRemoveLabel(
     w_windowID
     s_labelID | l_labelID
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Removes the specified labels (graph labels and marker labels) in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
s_labelID	ID of a graph or a marker label to be removed.
l_labelID	A list containing IDs of graph and marker labels to be removed.
?subwindow *x_subwindow*	Identification number of the subwindow (found in the top-right corner of the subwindow) from which graph and marker labels are to be removed. If you do not specify this argument, the labels are removed from the current subwindow.

Value Returned

t	Specified graph and marker labels are removed.
nil	The specified Waveform window, subwindow, or label ID does not exist.

Examples

The following example returns the trace numbers and names of the waveforms plotted in subwindow 2 of the specified Waveform window.

```
awvGetWaveNameList(window(3) ?subwindow 2)
=>
```

```
((1 2)
    ("net10" "out")
)
```

The following example places a marker label `markerLabel1` on the trace `out` at `x=150ns` and `y=3`.

```
awvPlaceWaveformLabel(window(3) 2 list(150ns 3) "markerLabel1" nil ?textOffset t
?subwindow 2)
=> ("pointMarker[1.1.1]")
```

The label ID of markerLabel1 is "`pointMarker[1.1.1]`".

The following example places a marker label `markerLabel2` on the trace `net10` at `x=50ns` and `y=1`.

```
awvPlaceWaveformLabel(window(3) 1 list(50ns 1) "markerLabel2" nil ?textOffset t
?subwindow 2)
=> ("pointMarker[1.1.2]")
```

The label ID of `markerLabel2` is "`pointMarker[1.1.2]`".

The following example places a graph label `graphLabel1` on trace `net10` at `x=1` and `y=1`.

```
awvPlaceWaveformLabel(window(3) 1 list(1 1) "graphLabel1" nil ?textOffset nil
?subwindow 2)
=> ("graphLabel[1.1.1]")
```

The label ID of `graphLabel1` is "`graphLabel[1.1.1]`".

The following example places a graph label `graphLabel2` on trace `out` at `x=0` and `y=1`.

```
awvPlaceWaveformLabel(window(3) 2 list(0 1) "graphLabel2" nil ?textOffset nil
?subwindow 2)
=> ("graphLabel[1.1.2]")
```

The label ID of graphLabel2 is "`graphLabel[1.1.2]`".

The following example removes `graphLabel2` from subwindow `2` of the specified Waveform window.

```
awvRemoveLabel(window(3) "graphLabel[1.1.2]" ?subwindow 2)
=> t
```

The following example removes `markerlabel1`, `markerLabel2`, and `graphLabel1` from subwindow `2` of the specified Waveform window.

```
awvRemoveLabel(window(3) list("pointMarker[1.1.2]" "pointMarker[1.1.1]"
"graphLabel[1.1.1]") ?subwindow 2)
=> t
```

awvRemoveSubwindowTitle

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

Description

Removes the title from a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow from which the title is to be removed. If you do not specify this argument, the title is removed from the current subwindow.

Value Returned

t	Title is removed from a subwindow of the specified Waveform window.
nil	The specified Waveform window or subwindow does not exist.

Examples

The following example removes the title from the subwindow 2 of the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:77
awvRemoveSubwindowTitle(win ?subwindow 2)
=> t
```

The following example removes the title from the current subwindow of the specified Waveform window.

```
awvRemoveSubwindowTitle(window(77))
```

```
=> t
```

awvRemoveTitle

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

Description

Removes the title from the specified Waveform window.

Arguments

w_windowID	Waveform window ID.

Value Returned

t	Title is removed from the specified Waveform window.
nil	The specified Waveform window does not exist.

Examples

The following example removes the title from the current Waveform window.

```
win = awvGetCurrentWindow()
=> window:77
awvRemoveTitle(win)
=> t
```

The following example removes the title from the specified Waveform window.

```
awvRemoveTitle(window(78))
=> t
```

awvResetAllWindows

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

Description

Resets all Waveform windows that are returned by `awvGetWindowList`.

Contents of the windows are erased and subwindows whose update statuses are turned on are deleted. To delete all subwindows, regardless of their update statuses, set the `?force` argument to `t`.

Waveform windows remain at their current sizes and locations.

Arguments

`?force` *g_force*	Specifies whether to delete all subwindows or only those subwindows whose update statuses are turned on. Valid values are: ■ `t`: All subwindows are deleted regardless of their update statuses. ■ `nil`: Only those subwindows whose update statuses are turned on are deleted. The default value is `nil`.

Value Returned

`t`	Waveform windows are reset successfully.

Examples

The following example returns IDs of the Waveform windows that are currently open.

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

The following example resets all Waveform windows and deletes only those subwindows whose update statuses are turned on.

```
awvResetAllWindows()
=> t
```

The following example resets all Waveform windows and deletes all subwindows, regardless of their update statuses.

```
awvResetAllWindows(?force t)
=> t
```

awvResetWindow

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

Description

Resets the specified Waveform window to the state of a new window.

Contents of the specified window are erased and subwindows whose update statuses are turned on are deleted. To delete all subwindows, regardless of their update statuses, set the `?force` argument to `t`.

The Waveform window remains at their current sizes and locations.

Arguments

w_windowID	Waveform window ID.
?force *g_force*	Specifies whether to delete all subwindows or only those subwindows whose update statuses are turned on. Valid values are: ■ `t`: All subwindows are deleted regardless of their update statuses. ■ `nil`: Only those subwindows whose update statuses are turned on are deleted. The default value is `nil`.

Value Returned

`t`	The specified Waveform window is reset successfully.
`nil`	The specified Waveform window does not exist.

Examples

The following example returns IDs of the Waveform windows that are currently open.

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

The following example resets the specified Waveform window and deletes only those subwindows whose update statuses are turned on.

```
awvResetWindow(window(4))
=> t
```

The following example resets the specified Waveform window and deletes all subwindows, regardless of their update statuses.

```
awvResetWindow(window(4) ?force t)
=> t
```

awvResumeViVA

```
awvResumeViVA(
     )
```

Description

Resumes Virtuoso Visualization and Analysis XL (if suspended) when called from CIW, provided that all the licensing requirements are met.

Arguments

None

Value Returned

None

Examples

None

awvRfLoadPull

```
awvRfLoadPull(
     w_waveform
     [ ?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 on two-dimensional sweep PSS results. The inner sweep must be `phase` and the outer sweep must be `mag`.

Arguments

w_waveform Waveform of the signal.

?maxValue *x_maxValue*

Largest value of the contour to be drawn.

Default value is `nil`, which indicates that the largest value is to be taken from the results.

?minValue *x_minValue*

Smallest value of the contour to be drawn.

Default value is `nil`, which indicates that the smallest value is to be taken from the results.

?numCont *x_numCont*

Number of points on the contour.

Default values is `8`.

?closeCont *g_closeCont*

	Specifies whether to draw a closed contour. Valid values are: ■ `t`: Draws a closed contour. ■ `nil`: Draws an open contour.
`?name` *`t_name`*	Name of the contour. Default value is `p`.

Value Returned

`t`	Load pull contour is drawn.
`nil`	Load pull contour cannot be drawn because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example opens the loadpull simulation results stored in the specified directory.

```
openResults("/home/user/loadpullsim/ExampleLibRF/lnaSimple/maestro/results/
maestro/loadPULL/1/ExampleLibRF_lnaSimple_1/psf")
=>
"/home/user/loadpullsim/ExampleLibRF/lnaSimple/maestro/results/maestro/loadPULL/
1/ExampleLibRF_lnaSimple_1/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
(hb_mt_fi hb_mt_fd model instance output
    designParamVals primitives subckts variables
)
```

The following example selects the `hb_mt_fi` results from the results directory.

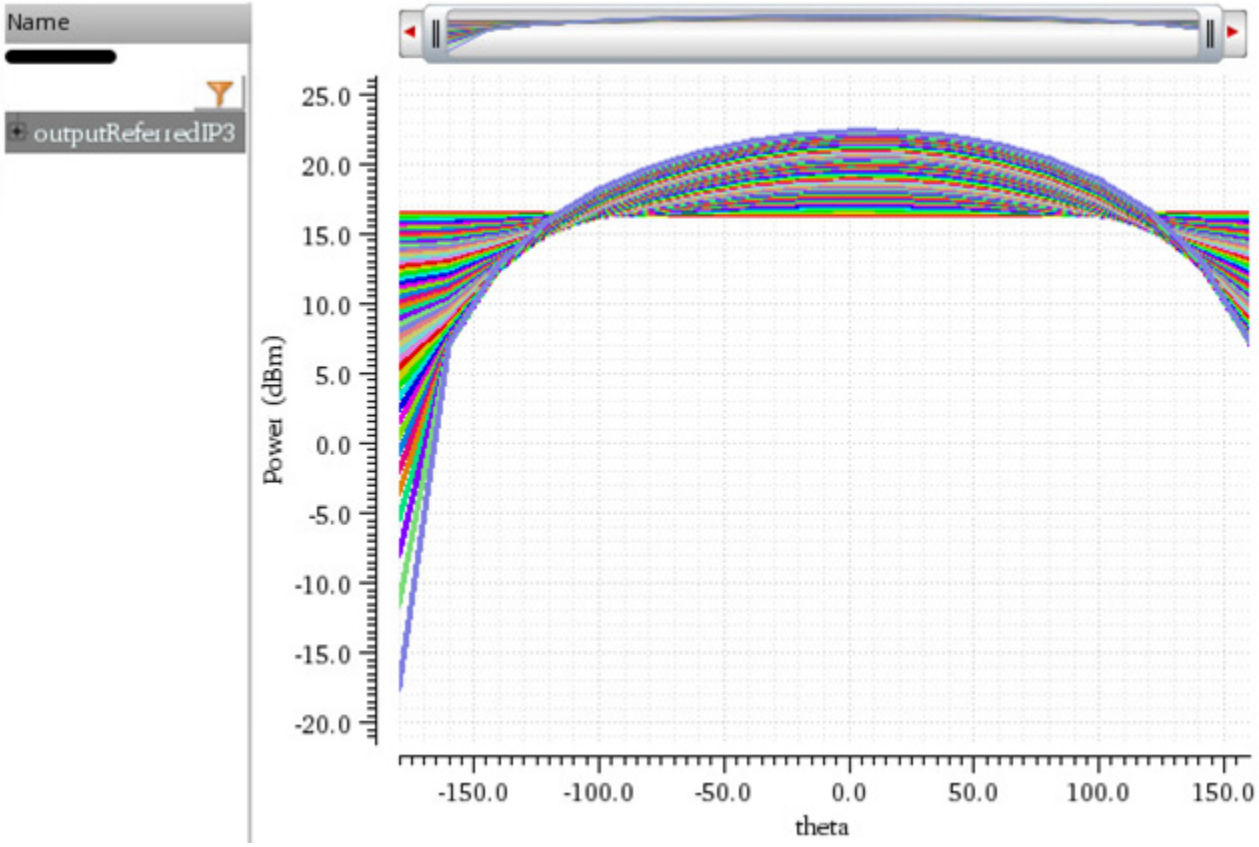
```
selectResults('hb_mt_fi)
=> stdobj@0x32b87cf8
```

The following example creates a waveform object `ip3`, which represents the third-order intercept point (IP3) measurement done using two-tone harmonic balance analysis (`hb_mt`).

```
ip3=ipnVRI((v("/net047" ?result "hb_mt_fi") - 0.0) '(2 -1) '(0 1) ?rport
resultParam("PORT0:r" ?result "hb_mt_fi") ?epoint -34 ?psweep nil ?measure
"Output")
=> srrWave:0x3868af50
```

The following example plots the waveform object `ip3` in the Waveform window `win1`.

```
awvPlotWaveform(
                win1
                list(ip3)
                ?expr list("outputReferredIP3")
                )
=> t
```



The following example creates another Waveform window and returns it window ID.

```
win2=awvCreatePlotWindow()
=> window:4
```

The following example sets display mode of the Waveform window `win2` to `smith`.

```
awvSetDisplayMode(win2 "smith")
```

```
=> t
```

The following example sets the Smith display of the Waveform window `win2` to `impedance`.

```
awvSetSmithModeType(win2 "impedance")
=> t
```

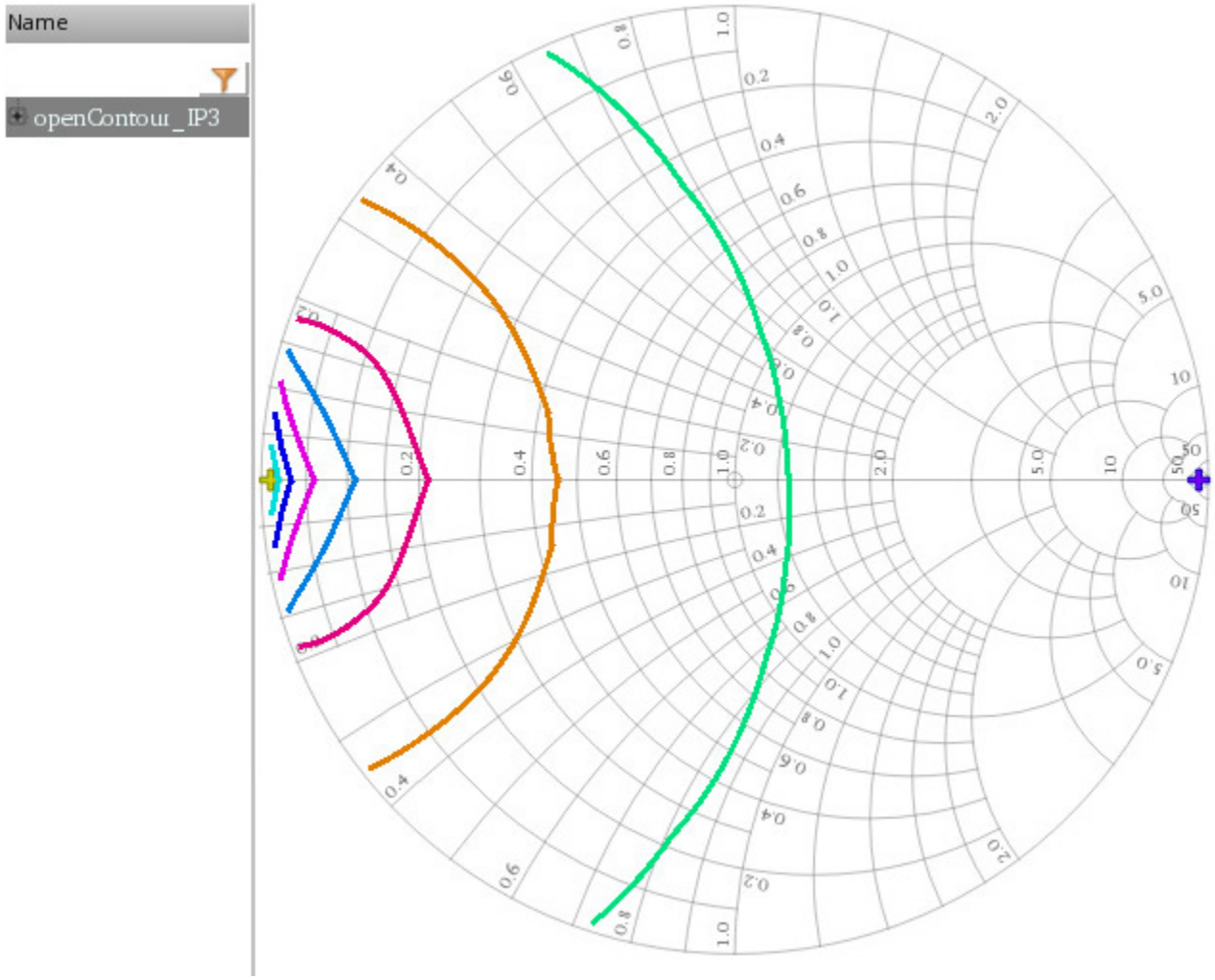
The following example creates a waveform object `openContour`, which represents `9` loadpull contours drawn for the input waveform `ip3`. In this example, note that the largest and smallest values of the contours are taken from the simulation results.

```
openContour=awvRfLoadPull(ip3 ?maxValue nil ?minValue nil ?numCont 9 ?closeCont
nil)
=> srrWave:0x3868d5f0
```

The following example plots the waveform object `openContour` in the Waveform window `win2`.

```
awvPlotWaveform(
                win2
                list(openContour)
                ?expr list("openContour_IP3")
                )
```

```
=> t
```



The following example creates another Waveform window and returns it window ID.

```
win3=awvCreatePlotWindow()
=> window:5
```

The following example sets display mode of the Waveform window `win3` to `smith`.

```
awvSetDisplayMode(win3 "smith")
=> t
```

The following example sets the Smith display of the Waveform window `win3` to `impedance`.

```
awvSetSmithModeType(win3 "impedance")
=> t
```

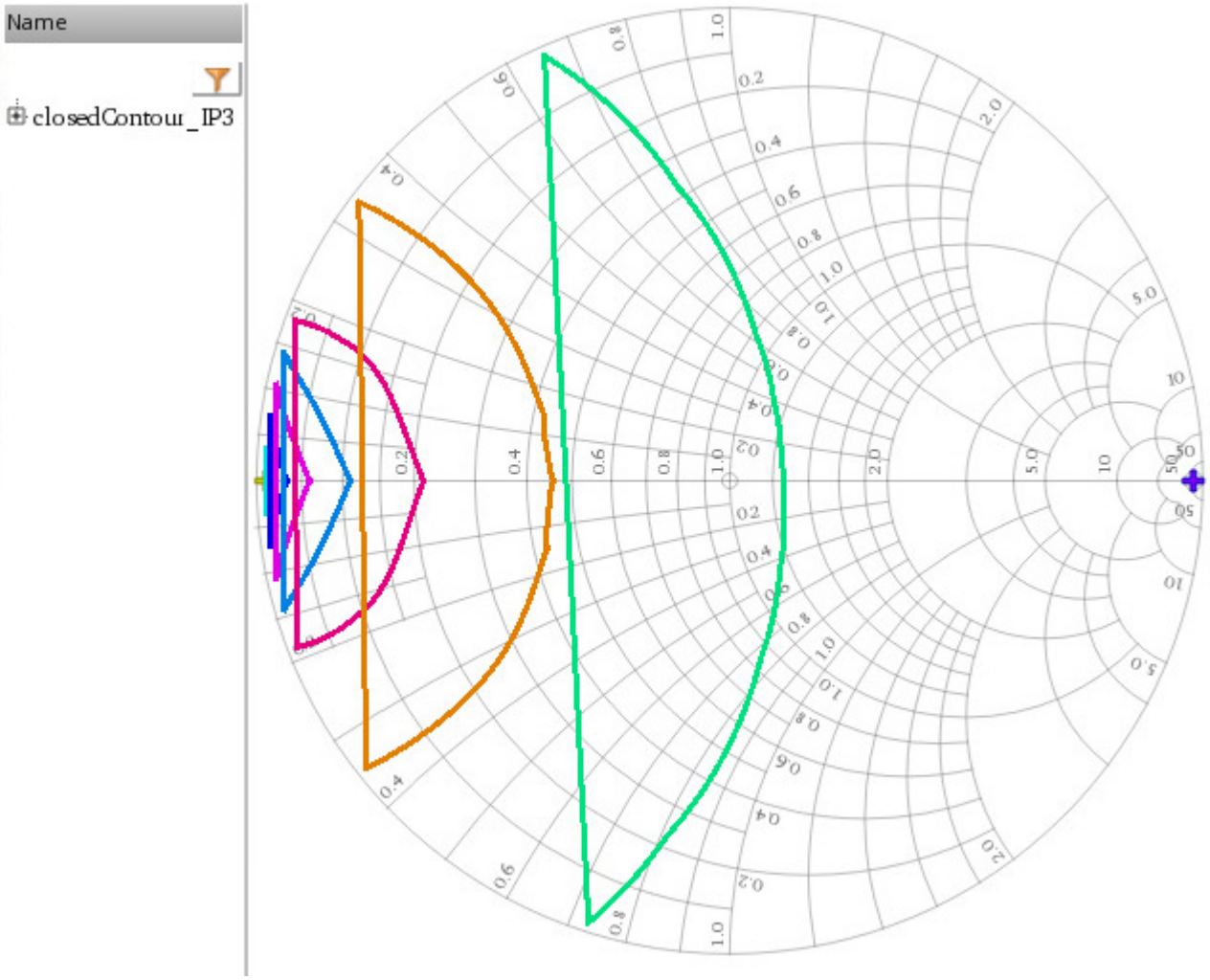
The following example creates a waveform object `closedContour`, which represents `9` loadpull contours drawn for the input waveform `ip3`. In this example, note that the largest and smallest values of the contours are taken from the simulation results.

```
closedContour=awvRfLoadPull(ip3 ?maxValue nil ?minValue nil ?numCont 9 ?closeCont
t)
=> srrWave:0x3866d700
```

The following example plots the waveform object `closedContour` in the Waveform window `win3`.

```
awvPlotWaveform(
                win3
                list(closedContour)
                ?expr list("closedContour_IP3")
                )
```

```
=> t
```



awvSaveMenuCB

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

Description

Displays the *Save* menu (*Windows* – *Save* ...). The function is defined in `dfII/etc/context/awv.cxt`.

Arguments

None

Value Returned

`t`	Save menu is successfully displayed.
`nil`	Save menu cannot be displayed because of an error.

Examples

The following example shows how to run the `awvSaveMenuCB` function.

```
awvSaveMenuCB()
=> nil
```

awvSaveToCSV

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

Description

Saves waveform data to the specified CSV file.

Arguments

l_waveforms	Waveform or a list of waveforms whose data is to be saved in the CSV file.
t_fileName	Name of the CSV file to which you want to save waveform data.
?from *x_from*	x-axis value after which you want to save waveform data.
?to *x_to*	x-axis value upto which you want to save waveform data.
?precision *x_precision*	Precision value to be used.
?step *x_step*	Step value to be used.
?linLog *t_linLog*	Specifies whether to save waveform data in logarithmic format. Valid values are: ■ `linear`: Waveform data is saved in linear format. ■ `log`: Waveform data is saved in logarithmic format. Default value is `linear`.
?exprList *l_expressionList*	List of expression names to be printed in the CSV file.
?mergeXAxis *g_mergeXAxis*	

Specifies whether to merge the x-axis values for all waveforms into a single column in the CSV file.

Valid values are:

- `t`: Merges x-axis values in a single column.
- `nil`: X-axis values are printed in separate columns.

Value Returned

`t`	Waveform data is saved to the CSV file.
`nil`	Waveform data cannot be saved because of an error.

Examples

The following example creates a Waveform window and returns the ID of the Waveform window.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified results directory `ampsim.raw`.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

The following example returns ID of the waveform of the signal `out`, which is available in the `tran-tran` result of the results directory `ampsim.raw`.

```
w1=v("out" ?result "tran-tran")
=> srrWave:0x36368020
```

The following example returns ID of the waveform that is obtained after applying the `flip` function to the signal `out`, which is available in the `tran-tran` result of the results directory `ampsim.raw`.

```
w2=flip(v("out" ?result "tran-tran"))
=> srrWave:0x36368040
```

The following example passes waveform objects `w1` and `w2` into a list named `waves`.

```
waves=list(w1 w2)
=> (srrWave:0x36368020 srrWave:0x36368040)
```

The following example saves waveform data for both waveforms `w1` and `w2` available in the list `waves` to a CSV file, `WaveData.csv`. The CSV file is saved under the `/home/user/` directory.

```
awvSaveToCSV(waves "/home/user/WaveData")
=> t
```

If you do not specify the path to the CSV file, the file is saved in the current working directory.

The following example saves waveform data for `w1` from `50ns` to `200ns` with a step value `10`.

Waveform data is saved in `logarithmic` format. The `dataLog.csv` file is saved in the current working directory.

```
awvSaveToCSV(w1 "dataLog" ?from 50ns ?to 200ns ?step 10 ?linLog "log")
=> t
```

The following example saves waveform data for `w1` from `50ns` to `200ns` with a step value `10`.

Waveform data is saved in `linear` format. The `dataLinear.csv` file is saved in the `/home/user/` directory.

```
awvSaveToCSV(w1 "/home/user/dataLinear" ?from 50ns ?to 200ns ?step 10 ?linLog
"linear")
=> t
```

The following examples create two waveform objects `w3` and `w4`

```
w3=drCreateWaveform(drCreateVec('double list(1 2 3 4 5 6)) drCreateVec('double
list(10 20 30 40 50 60)))
=> srrWave:0x36368050
w4=drCreateWaveform(drCreateVec('double list(1 2 3 4 5 6)) drCreateVec('double
list(5 10 15 20 25 30)))
=> srrWave:0x36368060
```

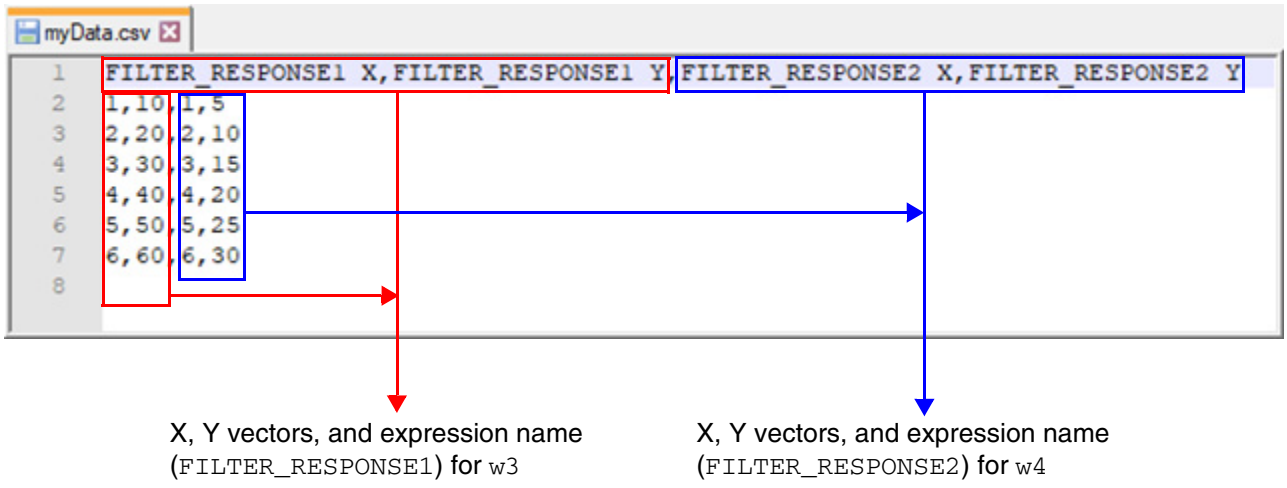
The following example passes waveform objects `w3` and `w4` into a list named `waveList`.

```
waveList=list(w3 w4)
=> (srrWave:0x36368050 srrWave:0x36368060)
```

The following example saves waveform data for the waveforms `w3` and `w4` contained in the list `waveList` to a CSV file, `myData.csv`. You can also specify expressions for these waveforms that you want print in the header of the CSV file.

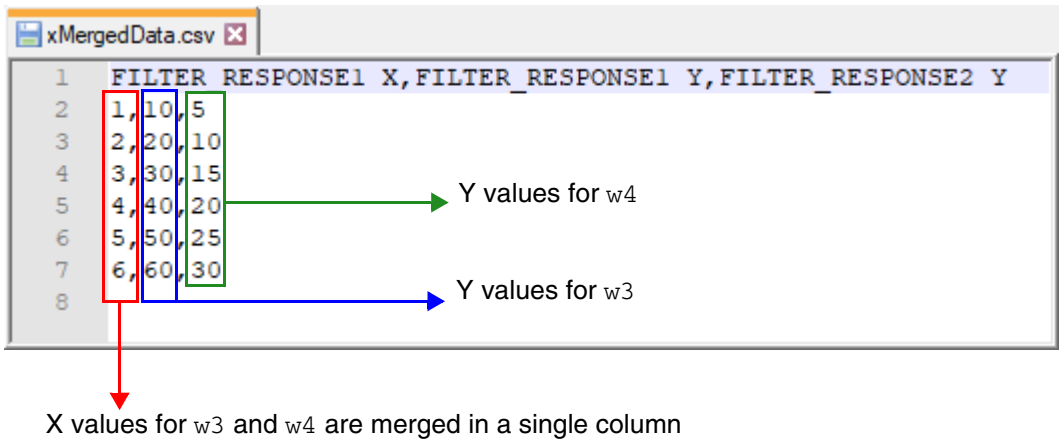
```
awvSaveToCSV(waveList "myData" ?exprList list("FILTER_RESPONSE1"
"FILTER_RESPONSE2"))
```

```
=> t
```



The following example saves waveform data for the waveforms `w3` and `w4` contained in the list `waveList` to a CSV file, `xMergedData.csv`. Note that x-vector values for both waveforms are merged in a single column in the CSV file.

```
awvSaveToCSV(waveList "xMergedData" ?exprList list("FILTER_RESPONSE1"
"FILTER_RESPONSE2") ?mergeXAxis t)
=> t
```



awvSaveWindow

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

Description

Saves the state of the specified Waveform window to an XML file with the `.grf` extension.

Arguments

w_windowID	Waveform window ID.
t_fileName	Path or name of the file in which the state of the specified Waveform window is to be saved. If you provide only a file name, the file is saved in the directory from which you started the software.

Value Returned

`t`	State of the Waveform window is saved successfully.
`nil`	State of the Waveform window cannot be saved because of an error.

Examples

The following example saves the state of the Waveform window `7` to `myFile` in the `/home/user` directory.

```
awvSaveWindow(window(7) "/home/user/myFile")
=> t
```

The following example saves the state of the Waveform window `7` to `myFile1` in the directory from which the software is started.

```
awvSaveWindow(window(7) "my_File1")
=> t
```

awvSaveWindowImage

```
awvSaveWindowImage(
     w_windowID
     t_path
     t_filePrefix
     g_cardLayout
     )
     => l_fileNames
```

Description

Saves the image of the specified Waveform window in a `.png` file.

Important

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

Arguments

w_windowID	Waveform window ID.
t_path	Path to the directory where the image of the window is to be saved.
t_filePrefix	Prefix you want to add to the default name of the file being saved. You can specify an empty string for this argument if you do not want to add any prefix to the default file name.
g_cardLayout	Specifies whether the Waveform window is in card layout mode. Valid values are: ■ `t`: The Waveform window is in card layout mode. ■ `nil`: The Waveform window is not in card layout mode.

Value Returned

l_fileNames	List containing the names of the files being saved.

Examples

The following example open the simulation results stored in the specified directory.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

The following example plots waveforms of signals `net10` and `out` in a Waveform window.

```
plot(v("net10" ?result "tran-tran") v("out" ?result "tran-tran"))
=> t
```

The following example returns ID of the Waveform window.

```
awvGetCurrentWindow()
=> window:3
```

The following example saves the Waveform window as a image file named `window:3.1.png`. Note that no prefix is added to the default name of the image file.

```
awvSaveWindowImage(window(3) "/home/user" "" t)
=> ("/home/user/window:3.1.png")
```

The following example saves the Waveform window as a image file named `TransientAnalysis:window:3.1.png`. Note that the specified prefix `TransientAnalysis:` is added to the default name of the image file.

```
awvSaveWindowImage(Window(3) "/home/user" "TransientAnalysis:" t)
=> ("/home/user/TransientAnalysis:window:3.1.png")
```

awvSetCurrentSubwindow

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

Description

Sets the specified subwindow as the current subwindow.

Arguments

w_windowID	Waveform window ID.
x_subwindow	Identification number of the subwindow (found in the top-right corner of the subwindow) that you want to set as the current subwindow.

Value Returned

`t`	The specified subwindow is set as the current subwindow.
`nil`	The specified Waveform window or the subwindow does not exist.

Examples

The subwindow `3` of the specified Waveform window is set as the current subwindow.

```
awvSetCurrentSubwindow(window(7) 3)
=> t
```

awvSetCurrentWindow

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

Description

Sets the specified window as the current Waveform window.

Arguments

w_windowID	ID of the Waveform window to be set as the current window.

Value Returned

`t`	The specified Waveform window is set as the current window.
`nil`	The specified Waveform window does not exist.

Examples

The following example sets the `window:4` as the current Waveform window.

```
awvSetCurrentWindow(window(4))
=> t
```

awvSetCursorPrompts

```
awvSetCursorPrompts(
     w_windowID
     x_yAxisNumber
     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_yAxisNumber	Number of y axis for which the tracking cursor prompt is to be set. Valid values are from `1` through `4`.
t_xPrompt	Prompt to set for the x-axis value.
t_yPrompt	Prompt to set for the y-axis value.
?stripNumber *x_stripNumber*	Number of the strip in which the tracking cursor prompt is to be set. Valid values are `1` through `20`. If you do not specify this argument, tracking cursor prompts are set in the current strip.
?subwindow *x_subwindow*	Number of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Tracking cursor prompt is set successfully.
`nil`	Tracking cursor prompt cannot be set because of an error.

Examples

The following example sets the tracking cursor prompt for

```
awvSetCursorPrompts(awvGetCurrentWindow() 1 "Time(ns)" "Voltage(V)")
=> t
```

awvSetDisplayMode

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

Description

Sets display mode of a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
t_mode	Display mode to be set for the subwindow. Valid values are: ■ `composite` ■ `smith` ■ `strip`
`?subwindow` *x_subwindow*	Identification number of the subwindow (found in the top-right corner of the subwindow) whose display mode is to be set. If you do not specify this argument, the specified display mode is set to the current subwindow.

Value Returned

`t`	Display mode of the subwindow is set successfully.
`nil`	The specified Waveform window, subwindow, or display mode does not exist.

Examples

The following example sets display mode of subwindow `2` of the current Waveform window to `smith`.

```
win = awvGetCurrentWindow()
=> window:89
awvSetDisplayMode(win "smith" ?subwindow 2)
=> t
```

The following example sets display mode of the current subwindow of the specified Waveform window to `strip`.

```
awvSetDisplayMode(window(90) "strip")
=> t
```

awvSetDisplayStatus

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

Description

Sets the display status of a subwindow in the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
g_enable	Specifies whether to turn on the display status of a subwindow. Valid values are: ■ `t`: Turns on the display status of a subwindow. ■ `nil`: Turns off the display status of a subwindow.
`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the specified display status is set for the current subwindow.

Value Returned

`t`	Display status of the subwindow is set successfully.
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following example turns on the display status of subwindow `1` in the current Waveform window.

```
awvSetDisplayStatus(awvGetCurrentWindow() t ?subwindow 1)
=> t
```

The following example turns off the display status of subwindow `2` in the current Waveform window.

```
awvSetDisplayStatus(awvGetCurrentWindow() nil ?subwindow 2)
=> t
```

awvSetInitializationTimeout

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

Description

Sets timeout period in second for ADE to establish connection with Virtuoso Visualization and Analysis XL. The default value of timeout period is `120` second, which means that ADE keeps trying to establish a connection with Virtuoso Visualization and Analysis XL for up to `120` second.

Arguments

x_timeOut	Timeout period in second.

Value Returned

x_timeOut	Timeout period in second.

Examples

The following example sets the timeout period to `240` second.

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

awvSetLegendPos

```
awvSetLegendPos(
     w_windowID
     t_legendPosition
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the position of the trace legend in a graph.

Arguments

w_windowID	Waveform window ID.
t_legendPosition	Position of the trace legend. Valid values are: ■ `above` ■ `inside` ■ `left`
`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the specified position of the trace legend is set to the graph in the current subwindow.

Value Returned

`t`	Position of the trace legend is set successfully.
`nil`	Position of the trace legend cannot be set because of an error.

Examples

The following example sets the position of the trace legend to `above` for the graph in subwindow `2` of the current Waveform window.

```
win=awvGetCurrentWindow()
=> window:10
awvSetLegendPos(win "above" ?subwindow 2)
=> t
```

The following example sets the position of the trace legend to `inside` for the graph in subwindow `3` of the specified Waveform window.

```
awvSetLegendPos(window(11) "inside" ?subwindow 3)
=> t
```

awvSetLegendWidth

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

Description

Sets the width of the trace legend in a subwindow of the specified Waveform window. The function works only when the trace legend position is set to `left`.

Arguments

w_windowID	Waveform window ID.
x_width	Width to be set for the trace legend.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, width of the trace legend is set in the current subwindow.

Value Returned

`t`	Width of the trace legend is set successfully.
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following example creates a Waveform window.

```
win=awvCreatePlotWindow()
=> window:10
```

Opens simulation results stored in the specified directory:

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

Plots the waveforms of `net10` and `out` signals from the simulation results.

```
awvPlotWaveform(
                win
                list(v("net10" ?result "tran-tran") v("out" ?result "tran-tran"))
                )
=> t
```

The following example sets the width of the trace legend to `240`.

```
awvSetLegendWidth(win 240)
=> t
```

The following example sets the width of trace legend to `120` in subwindow `2` of the specified Waveform window.

```
awvSetLegendWidth(window(9) 120 ?subwindow 2)
=> t
```

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.

Arguments

S_name	Name of the option to restore. The option name can be a string or a symbol.

Value Returned

`t`	The specified option is restored to its default value.
`nil`	The specified option cannot be restored to its default value because of an error.

Examples

The following example sets the `mode` option back to its default value, which is `composite`.

```
awvSetOptionDefault("mode")
=> t
```

Related Topics

Waveform Window Options

awvSetOptionValue

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.

Arguments

S_name	Name of the Waveform window option. The option name can be a string or a symbol.
g_value	Value to be set for the specified option.

Value Returned

g_value	Returns the new value of the option.
nil	Indicates an error.

Examples

The following example sets value of the `mode` option to `smith`.

```
awvSetOptionValue("mode" "smith")
=> "smith"
```

Related Topics

Waveform Window Options

awvSetOptionDefault

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 `composite` mode with only one y axis displayed.

Arguments

w_windowID	Waveform window ID.
l_origin	List of waveform coordinates that specify the new location for the axis origin.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, axis origin is set in the current subwindow.

Value Returned

`t`	Axis origin is set successfully.
`nil`	Axis origin cannot be set because of an error.

Examples

awvSetPlotStyle

```
awvSetPlotStyle(
     w_windowID
     t_plotStyle
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the plotting style in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
t_plotStyle	Plotting style to be set for the subwindow. Possible values are `auto`, `bar`, `scatterPlot`, `joined`, `polezero`, `histogram`, `point`, `line` and `spectral`.
`?subwindow` *x_subwindow*	Identification number of the subwindow (found in the top-right corner of the subwindow) whose plotting style is to be set. If you do not specify this argument, plotting style is set to the current subwindow.

Value Returned

`t`	Plotting style is set to the subwindow of the specified Waveform window.
`nil`	The specified Waveform window, subwindow, or the plotting style does not exist.

Examples

The following example sets the plotting style of subwindow `2` to `point` in the current Waveform window.

```
awvSetPlotStyle(awvGetCurrentWindow() "point" ?subwindow 2)
=> t
```

awvSetSmithModeType

```
awvSetSmithModeType(
     w_windowID
     t_type
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the type of Smith display to a subwindow in the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
t_type	Type of Smith display. Valid values are: ■ `polar` ■ `impedance` ■ `admittance` ■ `immittance`
`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the type of Smith display is set to the current subwindow.

Value Returned

`t`	Type of Smith display is set to the subwindow.
`nil`	Type of Smith display cannot be set because of an error.

Examples

The following examples sets the type of Smith display of subwindow `8` of the specified Waveform window to `immittance`.

```
awvSetSmithModeType(window(90) "immittance" ?subwindow 8)
=> t
```

The following example sets the type of Smith display of the current subwindow in the specified Waveform window to `polar`.

```
awvSetSmithModeType(window(91) "polar")
=> t
```

awvSetSmithXLimit

```
awvSetSmithXLimit(
     w_windowID
     l_minMax
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the x axis display limits for a subwindow with a Smith display mode.

Adjusts the horizontal limit of the graph to include the user-specified limit and positions the graph such that the midpoint of the user limit is centered. Depending on the aspect ratio of the window and current zoom state, the resulting limit may differ from the user-specified limit.

This command does not take effect if the display mode is set to strip or composite.

Arguments

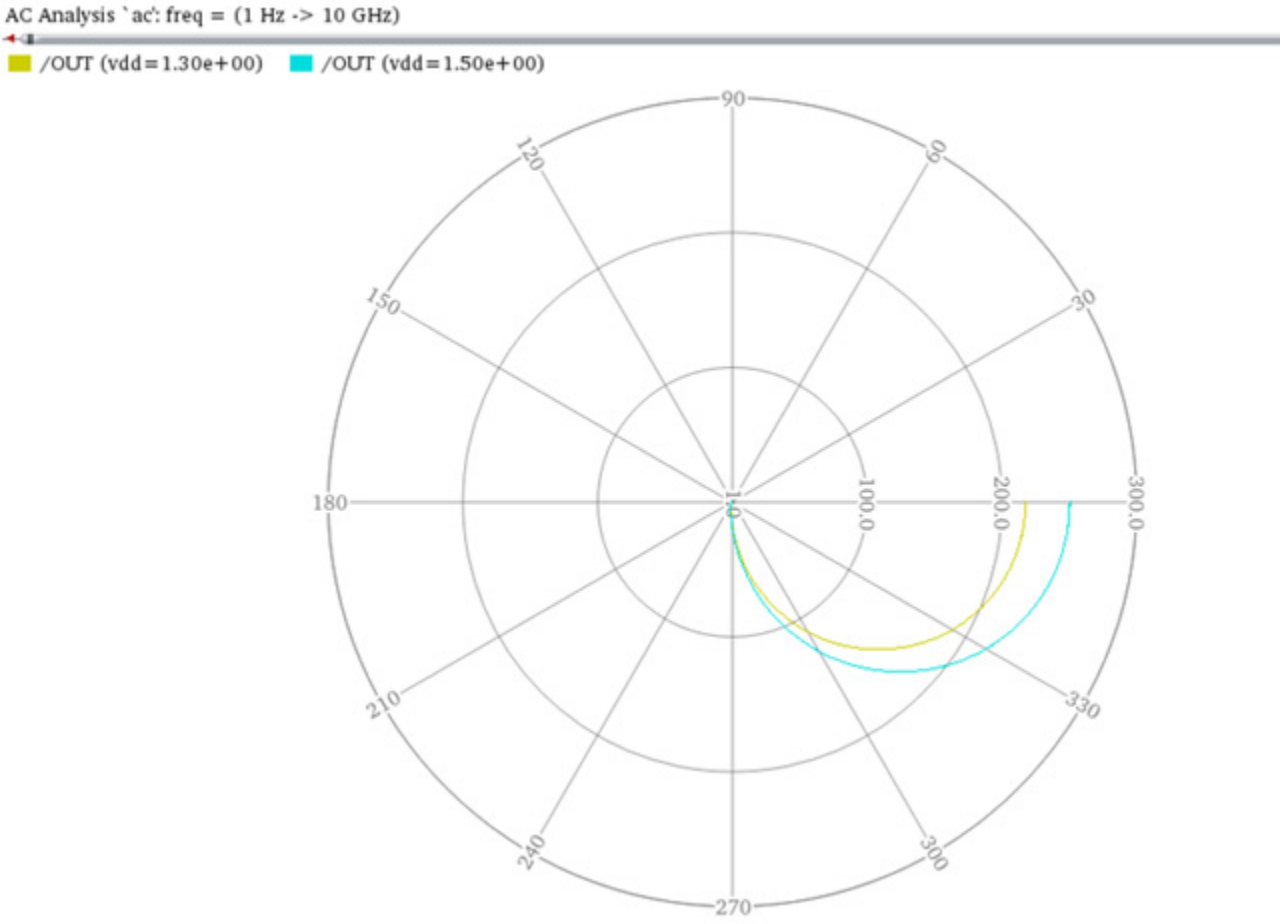
w_windowID	Waveform window ID.
l_minMax	List of two numbers in waveform coordinates that describe the limits for the display. The first number is the minimum and the second is the maximum. If this argument is set to nil, the limit is set to autoscale.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

t	Display limits for x axis are set successfully.
nil	Display limits cannot be set because of an error.

Examples

Consider the following graph window, where both X and Y limits are `[-300, 300]`.

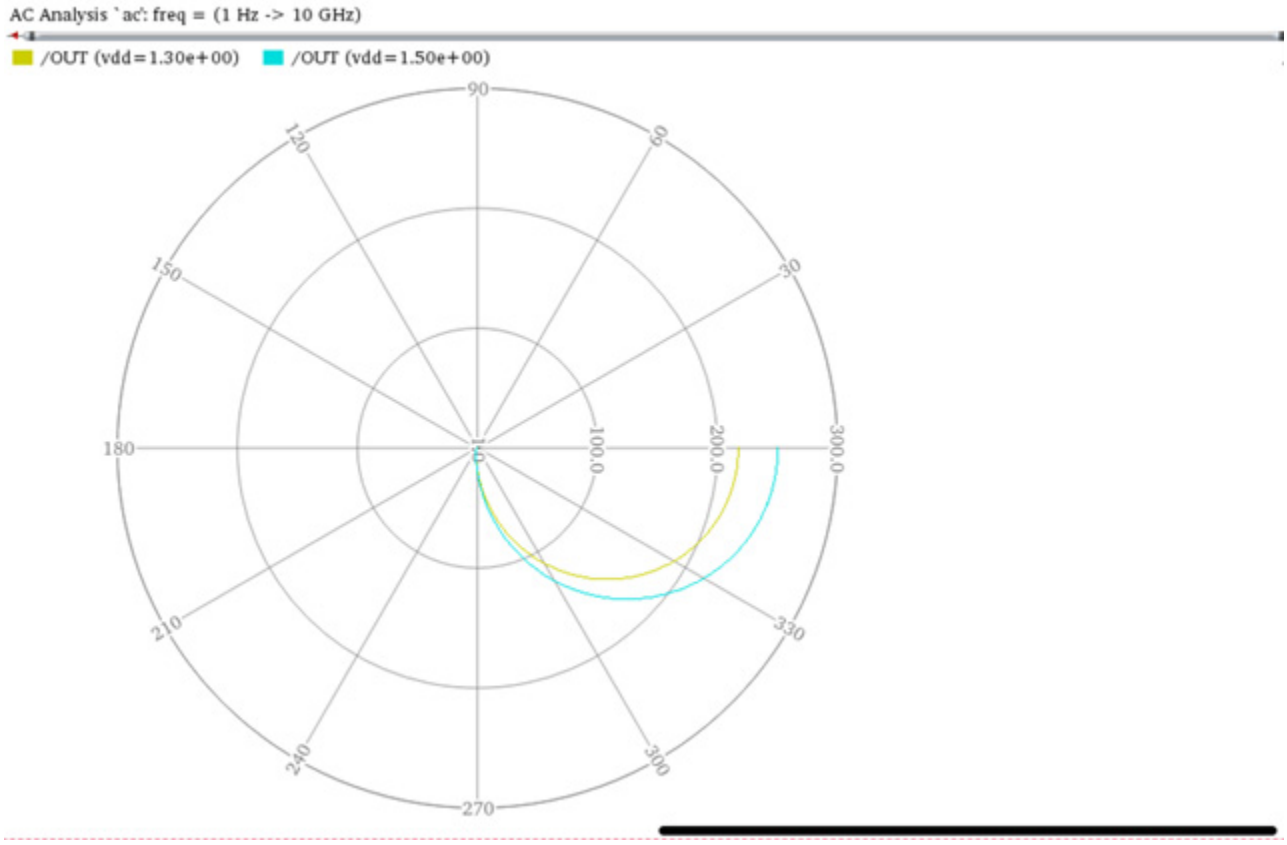


Run the following:

```
awvSetSmithXLimit(awvGetCurrentWindow() list(0 300) ?subwindow 1)
```

The x limit is now [`0, 300`] and the y limit remains [`-300, 300`].

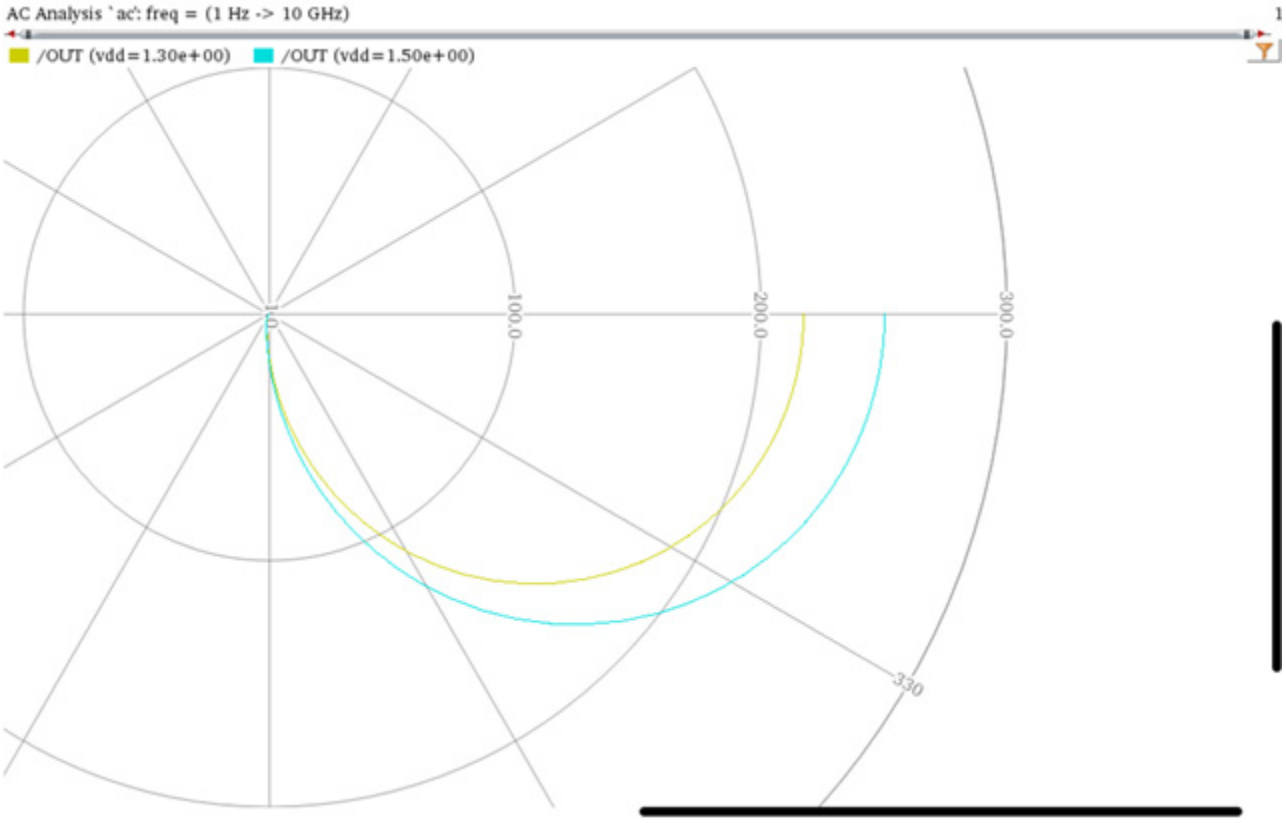
To preserve the y limit without stretching the graph, the horizontal zoom is unchanged, but the graph is adjusted such that [0, 300] is horizontally centered.



Now, run the following command:

```
awvSetSmithYLimit(awvGetCurrentWindow() list(-200 100) ?subwindow 1)
```

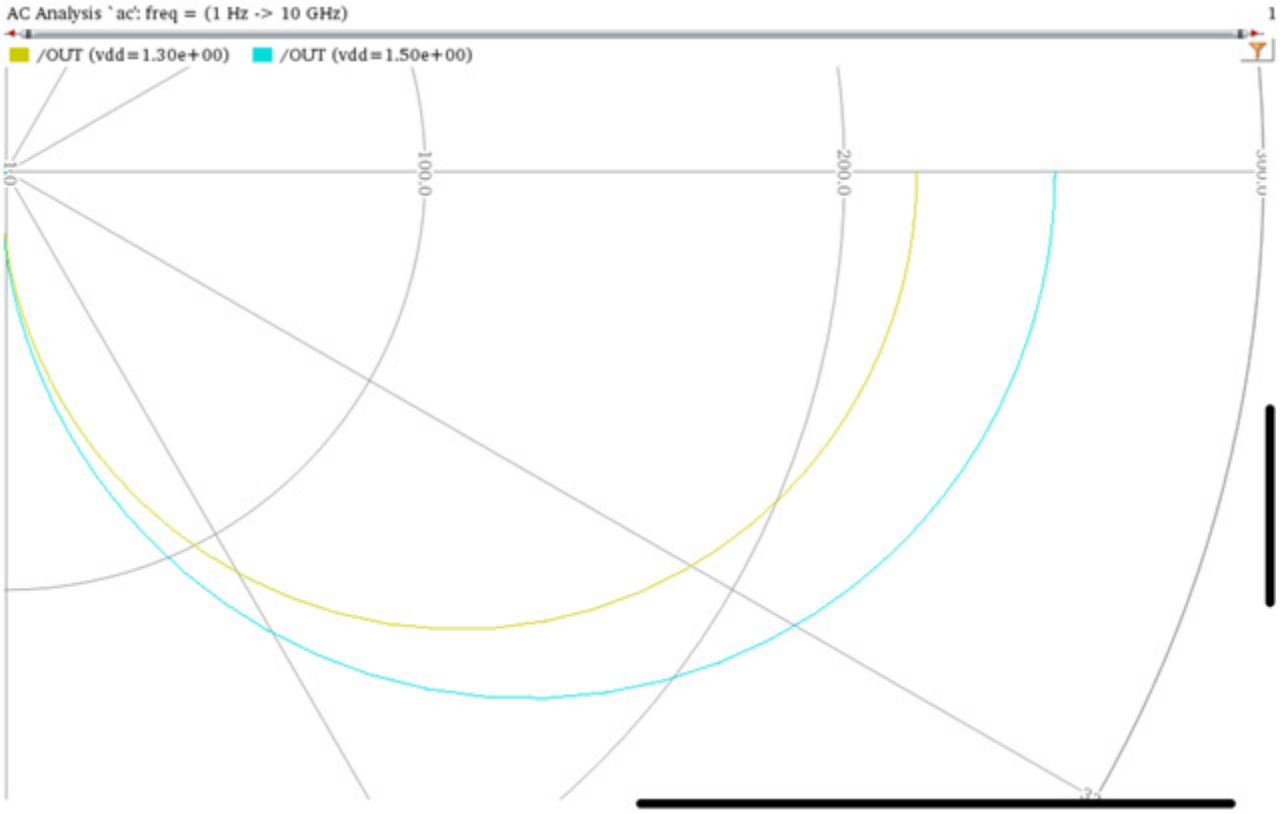
The x limit is still set to [0, 300] but the y limit is now [-200, 100]. The graph is vertically zoomed to [-200, 100], but the horizontal zoom is forced to include more than the user-specified x limit.



Now, when you run the following:

```
awvSetSmithYLimit(awvGetCurrentWindow() list(-150 25) ?subwindow 1)
```

The resulting limits match the user-specified limits only if the aspect ratio of the zoom rectangle and graph area match. In this example, setting x limit to [`0, 300`] and y limit to [`-150, 25`] fits this requirement.



awvSetSmithYLimit

```
awvSetSmithYLimit(
     w_windowID
     l_minMax
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the y axis display limits for a subwindow with a Smith display mode.

Adjusts the vertical limit of the graph to include the user-specified limit and positions the graph such that the midpoint of the user limit is centered. Depending on the aspect ratio of the window and current zoom state, the resulting limit may differ from the user-specified limit.

This command does not take effect if the display mode is set to strip or composite.

Arguments

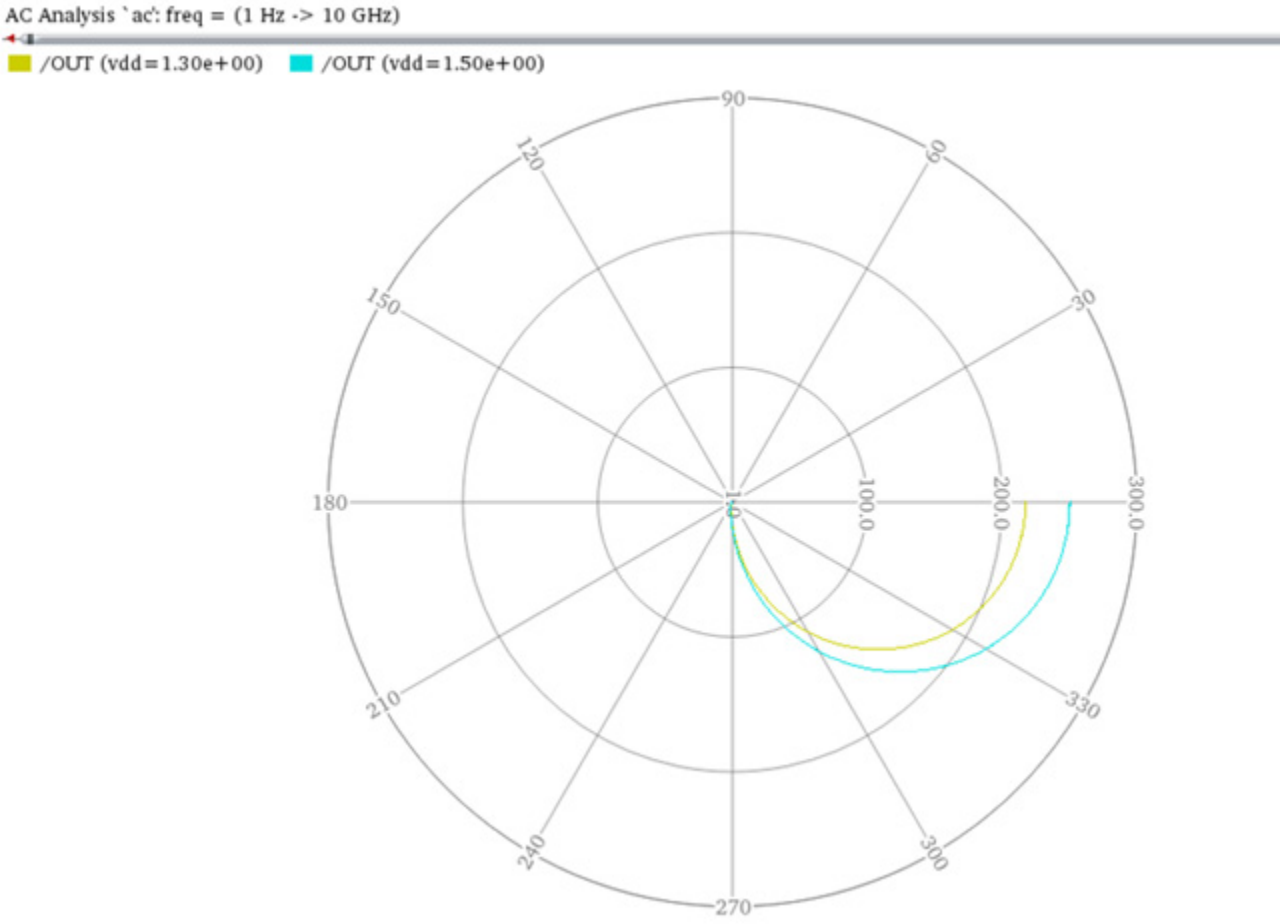
w_windowID	Waveform window ID.
l_minMax	List of two numbers in waveform coordinates that describe the limits for the display. The first number is the minimum and the second is the maximum. If this argument is set to `nil`, the limit is set to auto-scale.
`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Display limits for y axis are set successfully.
`nil`	Display limits cannot be set because of an error.

Examples

Consider the following graph window, where both X and Y limits are [-300, 300].

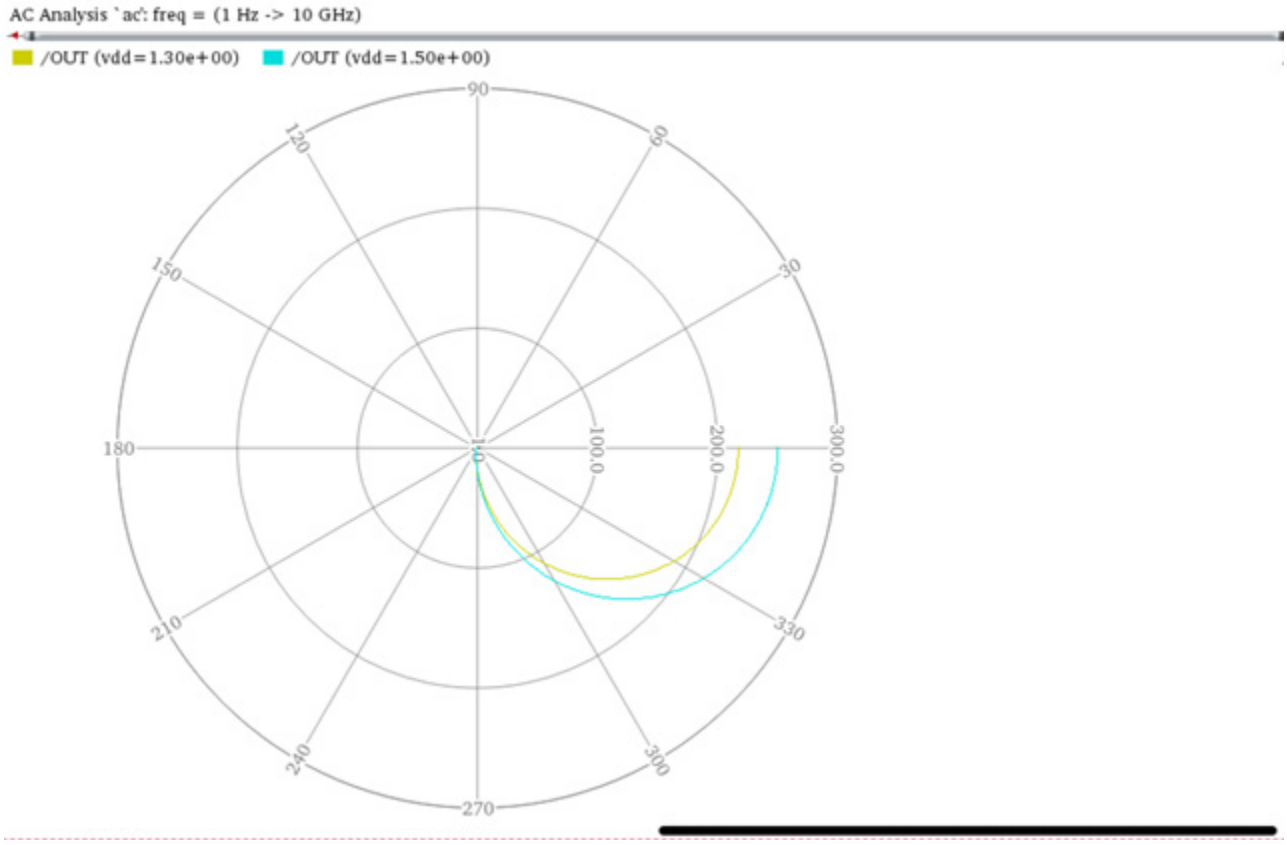


Run the following:

```
awvSetSmithXLimit(awvGetCurrentWindow() list(0 300) ?subwindow 1)
```

The x limit is now `[0, 300]` and the y limit remains `[-300, 300]`.

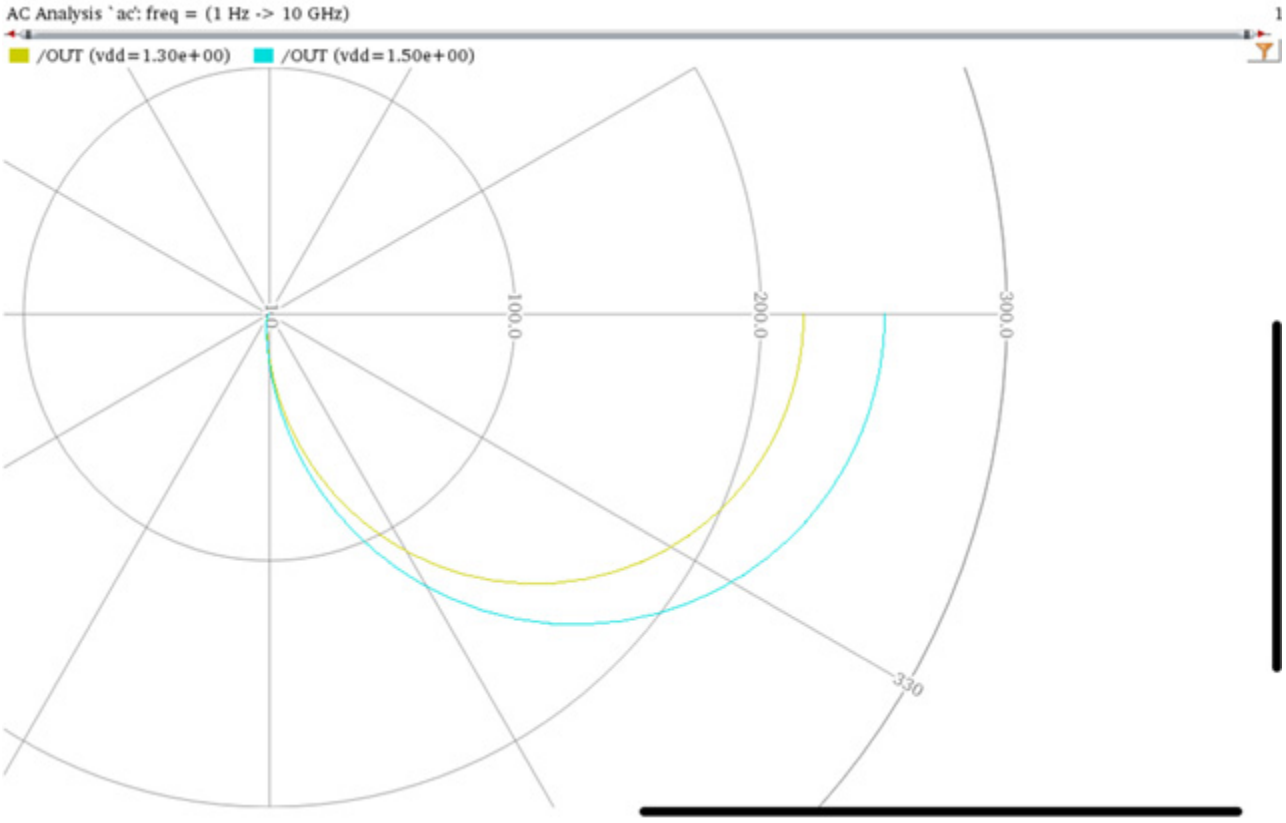
To preserve the y limit without stretching the graph, the horizontal zoom is unchanged, but the graph is adjusted such that `[0, 300]` is horizontally centered.



Now, run the following command:

```
awvSetSmithYLimit(awvGetCurrentWindow() list(-200 100) ?subwindow 1)
```

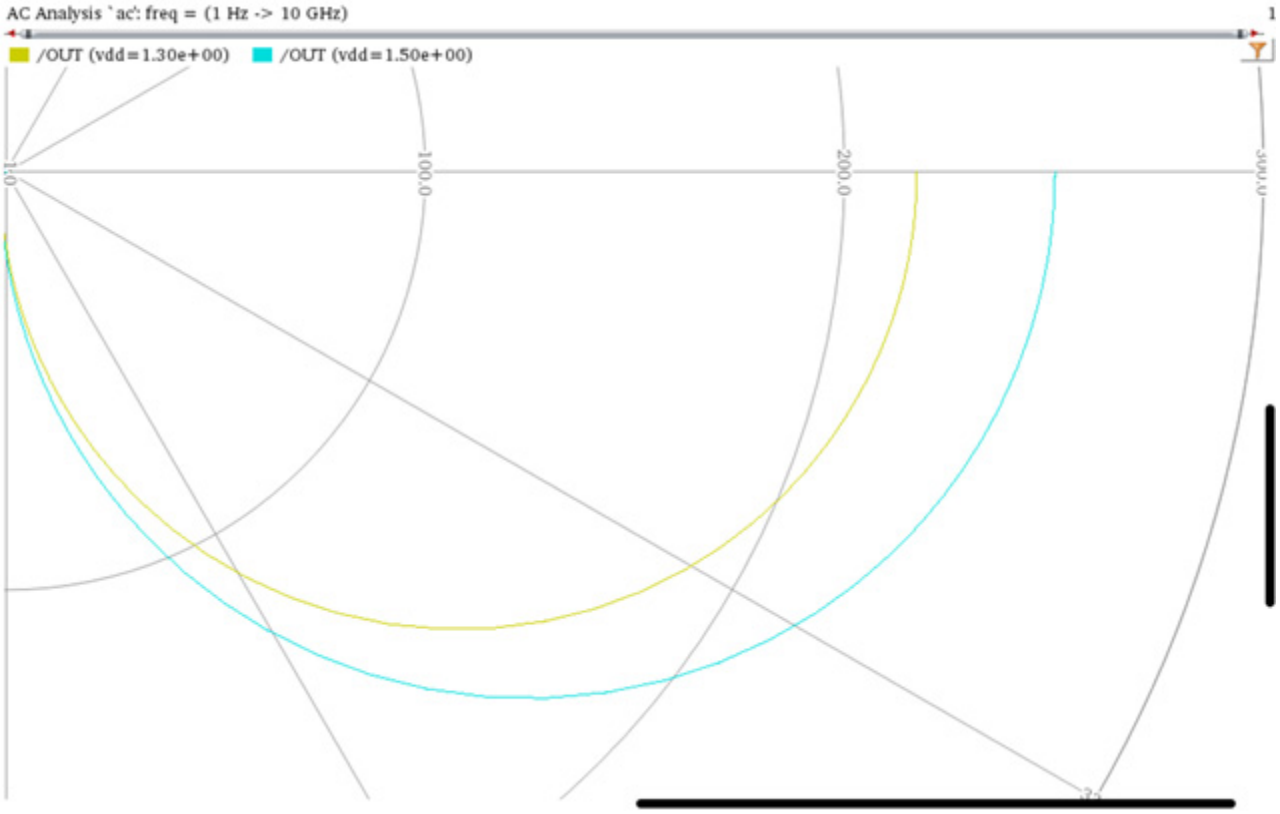
The x limit is still set to `[0, 300]` but the y limit is now `[-200, 100]`. The graph is vertically zoomed to `[-200, 100]`, but the horizontal zoom is forced to include more than the user-specified x limit.



Now, when you run the following:

```
awvSetSmithYLimit(awvGetCurrentWindow() list(-150 25) ?subwindow 1)
```

The resulting limits match the user-specified limits only if the aspect ratio of the zoom rectangle and graph area match. In this example, setting x limit to `[0, 300]` and y limit to `[-150, 25]` fits this requirement.



awvSetUpdateStatus

```
awvSetUpdateStatus(
     w_windowID
     g_enable
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the update status of a subwindow in the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
g_enable	Specifies whether to turn on the update status of a subwindow. Valid values are: ■ `t`: Turns on the update status of a subwindow. ■ `nil`: Turns off the update status of a subwindow. A lock icon is displayed in the top-left corner of the subwindow whose update status is turned off.
`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the specified update status is set for the current subwindow.

Value Returned

`t`	Update status of the subwindow is set successfully.
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following example turns on the update status of subwindow `1` in the current Waveform window.

```
awvSetUpdateStatus(awvGetCurrentWindow() t ?subwindow 1)
=> t
```

The following example turns off the update status of subwindow `2` in the current Waveform window.

```
awvSetUpdateStatus(awvGetCurrentWindow() nil ?subwindow 2)
=> t
```

awvSetWaveformDisplayStatus

```
awvSetWaveformDisplayStatus(
     w_windowID
     x_waveIndex
     g_enable
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Displays or hides a waveform in a subwindow of the specified Waveform window.

You can use this function to hide a waveform without deleting it.

Arguments

w_windowID	Waveform window ID.
x_waveIndex	Index number of the waveform to be hidden or displayed. You can use the `awvGetWaveNameList` function to return a list of index numbers of the waveforms plotted in a subwindow of the specified Waveform window.
g_enable	Specifies whether to display or hide the specified waveform. Valid values are: ■ `t`: Displays the waveform. ■ `nil`: Hides the waveform.
`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	The waveform is displayed or hidden successfully.

`nil`	The specified Waveform window, subwindow, or the waveform index does not exist.

Examples

The following example returns the indexes and signal names of the waveforms plotted in the subwindow `2` of the current Waveform window.

```
win=awvGetCurrentWindow()
=> window:110
awvGetWaveNameList(win ?subwindow 2)
=>
((26 17 18 19)
    ("/net027" "/net028" "/net029" "/net030")
)
```

The following example hides the waveform of the signal `net030` plotted in the subwindow `2` of the current Waveform window.

```
awvSetWaveformDisplayStatus(win 19 nil ?subwindow 2)
=> t
```

awvSetXAxisLabel

```
awvSetXAxisLabel(
     w_windowID
     t_label
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Replaces the automatically computed label of the x axis with the specified label in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
t_label	Label to be displayed for the x axis.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, label for the x axis is set in the current subwindow.

Value Returned

t	Label for the x axis is set successfully.
nil	The specified Waveform window or subwindow does not exist.

Examples

The following example sets the label for the x axis to `Time in nanosecond` in the subwindow `2` of the current Waveform window.

```
win=awvGetCurrentWindow()
=> window:11
awvSetXAxisLabel(win "Time in nanosecond" ?subwindow 2)
=> t
```

The following example sets the label for the x axis to `Time in second` in the current subwindow of the specified Waveform window.

```
awvSetXAxisLabel(window(12) "Time in second")
=> t
```

awvSetXAxisMajorDivisions

```
awvSetXAxisMajorDivisions(
     w_windowID
     x_numMajorDiv
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the number of major divisions on the x axis of a graph in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
x_numMajorDiv	Number of major divisions to be set on the x axis.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

t	Number of the major divisions on the x axis is set successfully.
nil	The specified Waveform window or subwindow does not exist.

Examples

The following example sets the number of major divisions on the x axis to 20 in the current subwindow of the current Waveform window.

```
awvSetXAxisMajorDivisions(awvGetCurrentWindow() 20 ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
```

The following example sets the number of major divisions on the x axis to 30 in the subwindow 2 of the specified Waveform window.

```
awvSetXAxisMajorDivisions(window(16) 30 ?subwindow 2)
=> t
```

awvSetXAxisMinorDivisions

```
awvSetXAxisMinorDivisions(
     w_windowID
     x_numMinorDiv
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the number of minor divisions on the x axis of a graph in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
x_numMinorDiv	Number of minor divisions to be set on the x axis.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

t	Number of the minor divisions on the x axis is set successfully.
nil	The specified Waveform window or subwindow does not exist.

Examples

The following example sets the number of minor divisions on the x axis to 10 in the current subwindow of the current Waveform window.

```
awvSetXAxisMinorDivisions(awvGetCurrentWindow() 10 ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
```

The following example sets the number of minor divisions on the x axis to 20 in the subwindow 2 of the specified Waveform window.

```
awvSetXAxisMinorDivisions(window(16) 20 ?subwindow 2)
=> t
```

awvSetXAxisStepValue

```
awvSetXAxisStepValue(
     w_windowID
     x_stepSize
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the step size for the major divisions of x axis in the specified graph. The step value indicates the spacing between major divisions of x axis.

Arguments

w_windowID	Waveform window ID.
x_stepSize	Step size for the major divisions of x axis in the specified graph.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

t	Step size for the major divisions of x axis is set successfully.
nil	Either the specified step size is not a valid number or the specified Waveform window or subwindow does not exist.

Examples

The following examples set the step size for the major divisions of x axis to `100ns` in the subwindow `2` of the current Waveform window.

```
awvSetXAxisStepValue(awvGetCurrentWindow() 100n ?subwindow 2)
=> t
awvSetXAxisStepValue(awvGetCurrentWindow() 100ns ?subwindow 2)
=> t
```

```
awvSetXAxisStepValue(awvGetCurrentWindow() 1e-7 ?subwindow 2)
=> t
```

awvSetXAxisUseStepValue

```
awvSetXAxisUseStepValue(
     w_windowID
     g_useStepSize
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Specifies whether to use the step size for the major divisions of x axis in the specified graph.

Arguments

w_windowID	Waveform window ID.
g_useStepSize	Specifies whether to use the step size for the major divisions of x axis. Valid values are: ■ `t`: Step size is used. ■ `nil`: Step size is not used.
`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	The functions runs successfully.
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following example enables the use of step size for the major divisions of x axis in the subwindow `2` of the specified Waveform window.

```
awvSetXAxisUseStepValue(window(14) t ?subwindow 2)
```

```
=> t
```

The following example disables the use of step size for the major divisions of x axis in the subwindow 3 of the specified Waveform window

```
awvSetXAxisUseStepValue(window(15) nil ?subwindow 3)
=> t
```

awvSetXLimit

```
awvSetXLimit(
     w_windowID
     l_minMax
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the display limits of x axis in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
l_minMax	List of two x-axis values that define the range of display. The first value in the list sets the minimum value of x axis. The second value in the list sets the maximum value of x axis. If this argument is set to `nil`, the limit is set to `autoscale`.
`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Display limits of x axis are set successfully.
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following example sets the display limit of x axis from `20ns` to `100ns` in the current subwindow of the specified Waveform window.

```
awvSetXLimit(window(38) list(20ns 100ns))
=> t
```

The following example sets the display limits of x axis from `40ns` to `200ns` in the subwindow number `3` of the current Waveform window.

```
awvSetXLimit(awvGetCurrentWindow() list(40ns 200ns) ?subwindow 3)
=> t
```

awvSetXScale

```
awvSetXScale(
     w_windowID
     t_scale
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets display mode of x axis in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
t_scale	Scale to be set for the x axis. Valid values are `linear`, `log`, and `auto`.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Display mode of x axis is set successfully.
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following example sets display mode of x axis to `linear` in the subwindow `3` of the current Waveform window.

```
awvSetXScale(awvGetCurrentWindow() "linear" ?subwindow 3)
=> t
```

The following example sets display mode of x axis to `logarithmic` in the current subwindow of the specified Waveform window.

```
awvSetXScale(window(38) "log")
=> t
```

awvSetYAxisLabel

```
awvSetYAxisLabel(
     w_windowID
     x_yNumber
     t_label
     [ ?subwindow x_subwindow ]
     [ ?stripNumber x_stripNumber ]
     )
     => t / nil
```

Description

Sets the specified label on the specified y-axis number in the current strip or the specified strip.

Arguments

w_windowID	Waveform window ID.
x_yNumber	Number of y axis whose label is to be set. The value can be any integer from `1` through `4`.
t_label	Label to be set on y axis.
`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, label for the y axis is set in the current subwindow.
`?stripNumber` *x_stripNumber*	Strip number of the trace. If you do not specify this argument, the currently selected strip is used. If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.

Value Returned

`t`	Y-axis label is set successfully.

`nil`	The specified Waveform window, subwindow, y-axis number, or the strip number does not exist.

Examples

The following example sets the label `Signal Voltage` to the y-axis number `1` in the strip number `2` of subwindow `3` in the current Waveform window.

```
awvSetYAxisLabel(awvGetCurrentWindow() 1 "Signal Voltage" ?stripNumber 2
?subwindow 3)
=> t
```

awvSetYAxisMajorDivisions

```
awvSetYAxisMajorDivisions(
     w_windowID
     x_yNumber
     x_numMajorDiv
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the number of major divisions on the scale of the specified y axis of a strip in the specified subwindow of a Waveform window.

Arguments

w_windowID Waveform window ID.

x_yNumber Number of y axis for which the number of major divisions is to be set. The value can be any integer from `1` through `4`.

x_numMajorDiv Number of major divisions to be set on the scale of the specified y axis.

?stripNumber *x_stripNumber*

Strip number of the trace. If you do not specify this argument, the currently selected strip is used.

If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.

?subwindow *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Number of the major divisions on the scale of the specified y axis is set in the specified graph.
`nil`	The specified Waveform window, subwindow, y-axis number, or the strip number does not exist.

Examples

The following example sets the number of major divisions on the y-axis number `1` to `10` in the strip number `3` of the current subwindow in the current Waveform window.

```
awvSetYAxisMajorDivisions(awvGetCurrentWindow() 1 10 ?stripNumber 3 ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
```

The following example sets the number of major divisions on the y-axis number `1` to `15` in the strip number `2` in the subwindow `3` of the specified Waveform window.

```
awvSetYAxisMajorDivisions(window(11) 1 15 ?stripNumber 2 ?subwindow 3)
=> t
```

awvSetYAxisMinorDivisions

```
awvSetYAxisMinorDivisions(
     w_windowID
     x_yNumber
     x_numMinorDivisions
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the number of minor divisions on the scale of the specified y axis of a strip in the specified subwindow of a Waveform window.

Arguments

w_windowID Waveform window ID.

x_yNumber Number of y axis for which the number of minor divisions is to be set. The value can be any integer from `1` through `4`.

x_numMinorDivisions

Number of minor divisions to be set on the scale of the specified y axis.

?stripNumber *x_stripNumber*

Strip number of the trace. If you do not specify this argument, the currently selected strip is used.

If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.

?subwindow *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Number of the minor divisions on the scale of the specified y axis is set successfully.
`nil`	The specified Waveform window, subwindow, y-axis number, or the strip number does not exist.

Examples

The following example sets the number of minor divisions on the y-axis number `1` to `10` in the strip number `3` of the current subwindow in the current Waveform window.

```
awvSetYAxisMinorDivisions(awvGetCurrentWindow() 1 10 ?stripNumber 3 ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
```

The following example sets the number of minor divisions on the y-axis number `1` to `5` in the strip number `2` in the subwindow `3` of the specified Waveform window.

```
awvSetYAxisMinorDivisions(window(11) 1 5 ?stripNumber 2 ?subwindow 3)
=> t
```

awvSetYAxisStepValue

```
awvSetYAxisStepValue(
     w_windowID
     x_yNumber
     x_stepSize
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the step size for the major divisions of the specified y axis of a strip in the specified subwindow of a Waveform window. The step value indicates the spacing between major divisions of y axis.

Arguments

w_windowID	Waveform window ID.
x_yNumber	Number of y axis whose step size is to be returned. The value can be any integer from `1` through `4`.
x_stepSize	Step size for the major divisions of the specified y axis.
?stripNumber *x_stripNumber*	Strip number of the trace. If you do not specify this argument, the currently selected strip is used. If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Step size for the major divisions of the specified y axis is set successfully.
`nil`	The specified Waveform window, subwindow, y-axis number, or the strip number does not exist.

Examples

The following example sets the step size for the major divisions of the y-axis number `1` to `20mV` in the currently selected strip of subwindow `2` in the specified Waveform window.

```
awvSetYAxisStepValue(window(14) 1 20mV ?subwindow 2)
=> t
```

The following example sets the step size for the major divisions of the y-axis number `1` to `20mV` in the strip number `3` of subwindow `4` in the specified Waveform window.

```
awvSetYAxisStepValue(window(15) 1 2e-2 ?stripNumber 3 ?subwindow 4)
=> t
```

awvSetYAxisUseStepValue

```
awvGetYAxisUseStepValue(
     w_windowID
     x_yNumber
     g_useStepSize
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow x_subwindow ]
     )
     => l_infoList / nil
```

Description

Specifies whether to use the step size for the major divisions of the specified y axis in the specified graph.

Arguments

w_windowID — Waveform window ID.

x_yNumber — Number of y axis. The value can be any integer from `1` through `4`.

g_useStepSize — Specifies whether to use the step size for the major divisions of the specified y axis.

Valid values are:

- `t`: Step size is used.
- `nil`: Step size is not used.

`?stripNumber` *x_stripNumber*

Strip number of the trace. If you do not specify this argument, the currently selected strip is used.

If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.

`?subwindow` *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

l_infoList	A list containing the dependent axis number and the status of the step size set on the specified y-axis number.
`nil`	The specified Waveform window, subwindow, y-axis number, or the strip number does not exist.

Examples

The following example enables the use the of step size for the major divisions of the y-axis number `1` in the strip number `2` of the subwindow `3`.

```
awvSetYAxisUseStepValue(window(14) 1 t ?stripNumber 2 ?subwindow 3)
=>
(("depAxis[68.127.314]")
    (("useMajorDelta" "true"))
)
```

The following example disables the use the of step size for the major divisions of the y-axis number `1` in the strip number `3` of the subwindow `3`.

```
awvSetYAxisUseStepValue(window(14) 1 nil ?stripNumber 3 ?subwindow 3)
=>
(("depAxis[68.127.315]")
    (("useMajorDelta" "false"))
)
```

awvSetYLimit

```
awvSetYLimit(
     w_windowID
     x_yNumber
     l_minMax
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the display limits of the specified y axis for the waveform with the specified strip number in a subwindow. If you do not specify the `?stripNumber` argument, the limits are applied when the subwindow is in `composite` mode.

Arguments

w_windowID	Waveform window ID.
x_yNumber	Number of y axis whose display limits to be set. The value can be any integer from `1` through `4`.
l_minMax	List of two y-axis values that define the range of display. The first value in the list sets the minimum value of y axis. The second value in the list sets the maximum value of y axis. If this argument is set to `nil`, the limit is set to `autoscale`.
`?stripNumber` *x_stripNumber*	Identifies the strip in which limit of y axis is to be set. If you do not specify this argument, the currently selected strip is used. If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.
`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Display limits of the specified y-axis number are set successfully.
`nil`	The specified Waveform window, subwindow, y-axis number, or the strip number does not exist.

Examples

The following example sets the display limit of y-axis number `4` from `-400` to `400`. This setting takes effect only in `composite` mode.

```
awvSetYLimit(window(34) 4 list(-400 400))
=> t
```

The following example sets the display limit of y-axis number `2` in the strip number `4` of the current subwindow in the specified Waveform window. This setting takes effect only in `strip` mode.

```
awvSetYLimit(window(34) 2 list(-400 400) ?stripNumber 4)
=> t
```

awvSetYRange

```
awvSetYRange(
     w_windowID
     x_yNumber
     n_range
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Sets the range of y axis in the specified strip in a subwindow. The specified range must be positive.

The range is the difference between the maximum y-axis value and the minimum y-axis value. The maximum value for y axis is automatically computed according to the waveforms. The minimum is calculated by subtracting the specified range from the maximum y-axis value.

Arguments

w_windowID	Waveform window ID.
x_yNumber	Number of y axis whose range is to be set. The value can be any integer from `1` through `4`.
n_range	Range you want to set for y axis.

`?stripNumber` *x_stripNumber*

Identifies the strip in which range of y axis is to be set. If you do not specify this argument, the range is set in the current strip.

If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.

`?subwindow` *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Range for the specified y-axis is set successfully.
`nil`	The specified Waveform window, subwindow, y-axis number, or the strip number does not exist.

Examples

The following example sets the range of y-axis number `1` to `10` in strip number `2` of the current subwindow in the specified Waveform window.

```
awvSetYRange(window(3) 1 10 ?stripNumber 2)
=> t
```

awvSetWaveNameList

```
awvSetWaveNameList(
     l_traceNumbers
     l_traceNames
     )
     => t / nil
```

Description

Sets the names of the waveforms associated with the specified trace numbers.

You can use the `awvGetWaveNameList` function to return the name and numbers of the waveforms plotted in a subwindow of the specified Waveform window.

Arguments

l_traceNumbers	A list containing trace numbers of the waveforms whose names to be set. The number of items in the list *l_traceNumbers* must exactly match with those in the list *l_traceNames*.
l_traceNames	A list containing the waveform names to be set to the specified trace numbers. The number of items in the list *l_traceNumbers* must exactly match with those in the list *l_traceNames*.

Value Returned

`t`	A list containing numbers and names of the waveforms.
`nil`	Either none of the specified trace numbers exists or number of items in the lists *l_traceNumbers* and *l_traceNames* does not match.

Examples

The following example returns a list of trace numbers and names of the waveforms plotted in subwindow `4` of the current Waveform window.

```
awvGetWaveNameList(awvGetCurrentWindow() ?subwindow 4)
=>
```

```
((5 6 7)
    ("net35" "jitter" "out")
)
```

Waveform `net35`, `jitter`, and `out` waveforms are identified with trace numbers `5`, `6`, and `7` respectively.

The following example sets the name of the waveforms `jitter` (trace number `6`), `net35` (trace number `5`), and `out` (trace number `7`) to `wave6`, `wave5`, and `wave7`.

```
awvSetWaveNameList(list(list(6 5 7) list("wave6" "wave5" "wave7")))
=> t
```

awvSimplePlotExpression

```
awvSimplePlotExpression(
     w_windowID
     t_expr
     l_context
     g_replace
     [ ?expr l_exprList ]
     [ ?color l_colorList ]
     [ ?lineType l_lineTypeList ]
     [ ?lineStyle l_lineStyleList ]
     [ ?lineThickness l_lineThicknessList ]
     [ ?showSymbols l_showList ]
     [ ?dataSymbol l_symbolList ]
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Evaluates the *t_expr* expression and plots the resulting waveforms in a subwindow of the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
t_expr	String containing an expression that is evaluated when the command is issued and reevaluated at the completion of each simulation in auto-update mode.
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`.
g_replace	Specifies whether to overwrite the existing waveforms in the subwindow. Valid values are: ■ `t`: Existing waveforms are replaced by the resulting waveform evaluated by the expression *t_expr*. ■ `nil`: Existing waveforms are retained and the resulting waveform evaluated by the expression *t_expr* is appended in the subwindow.
`?expr` *l_exprList*	

List of waveform names to be displayed in the trace legend.

If you do not specify this argument the expression is displayed instead of waveform names.

`?color` *`l_colorList`*

List of colors for the waveforms. Available colors are defined in your technology file. Valid values are from `y1` through `y66`.

If you do not specify this argument, default colors are used.

`?lineType` *`l_lineTypeList`*

List specifying the type of line to be used for the waveforms.

Valid values are `line`, `bar`, `scatterPlot`, `poleZero`.

If you do not specify this argument, default value `line` is used.

`?lineStyle` *`l_lineStyleList`*

List specifying the line style to used for the waveforms.

Valid values are `solid`, `dash`, `dot`, `dashDot`, and `dashDotDot`.

If you do not specify this argument, default value `solid` is used.

`?lineThickness` *`l_lineThicknessList`*

List specifying the line thickness of the waveforms.

Valid values are `fine`, `medium`, `thick`, and `extraThick`.

If you do not specify this argument, default value `fine` is used.

`?showSymbols` *`l_showList`*

List of flags that specify whether to show the symbols on the waveforms. Valid values are `t` and `nil`.

The default value is `nil`, which means that symbols are not displayed on the waveforms.

The number of flags in the *`l_showList`* must match the number of symbols in the *`l_symbolList`*.

`?dataSymbol` *`l_symbolList`*

List of symbols to be displayed for data points on the waveforms.

To use a symbol, specify an integer or a single character corresponding to the symbol.

`?subwindow` *`x_subwindow`*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Expression *`t_expr`* is evaluated successfully and the resulting waveform is plotted in the subwindow of the specified Waveform window.
`nil`	Expression cannot be evaluated because of an error.

Examples

The following example creates a Waveform window and returns the ID of the Waveform window.

```
awvCreatePlotWindow()
=> window:3
```

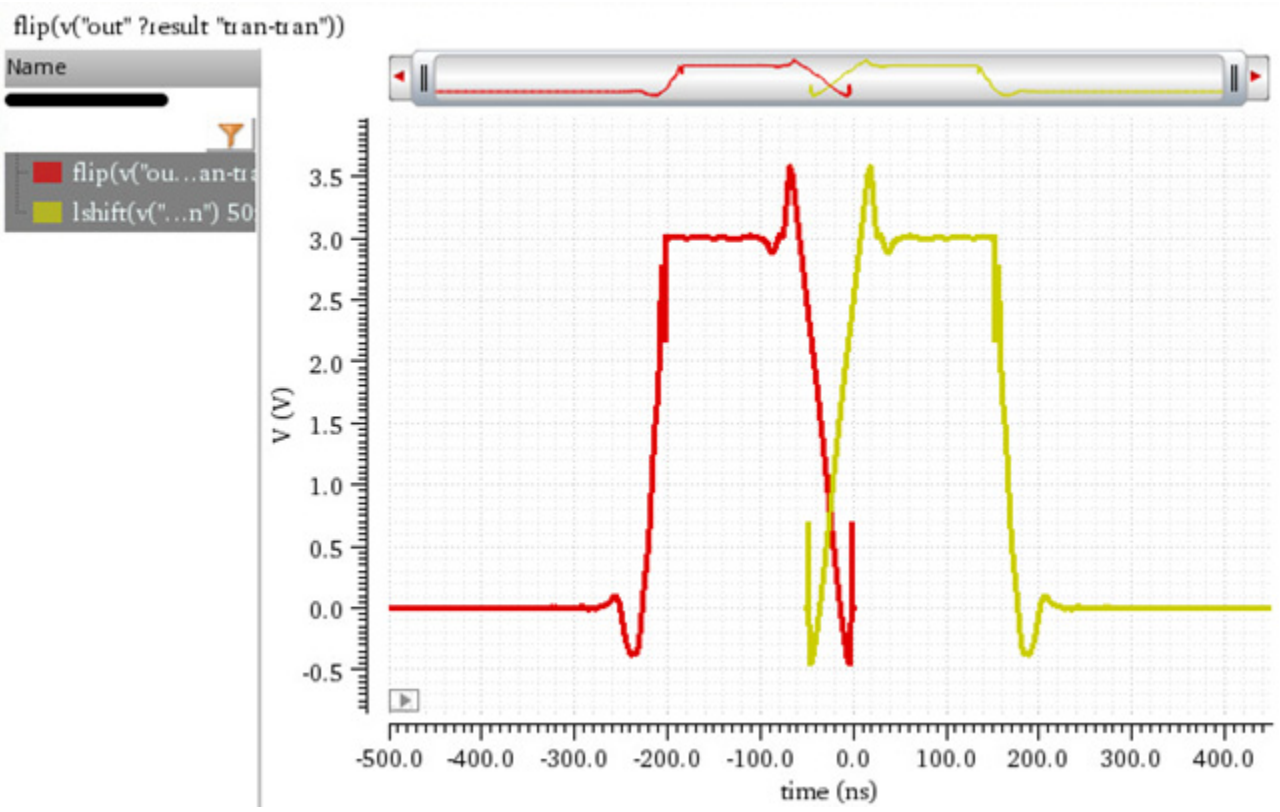
The following example opens simulation results stored in the specified directory `amsim.raw`.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

The following examples evaluate expressions `flip` and `lshift` applied on the signal `out`, which is available in the `tran-tran` result of the results directory `ampsim.raw`.

```
awvPlotSimpleExpression("flip(v(\"out\" ?result \"tran-tran\"))" ?plotStyle
"Append" ?graphType "Rectangular" ?graphModifier "Magnitude")
=> t
awvPlotSimpleExpression("lshift(v(\"out\" ?result \"tran-tran\") 50ns )"
?plotStyle "Append" ?graphType "Rectangular" ?graphModifier "Magnitude")
=> t
```

The resulting waveforms evaluated from the expressions are plotted in `append` mode in the Waveform window created by the function `awvCreatePlotWindow`.



The following example returns the index numbers and names of the waveforms plotted in the current Waveform window.

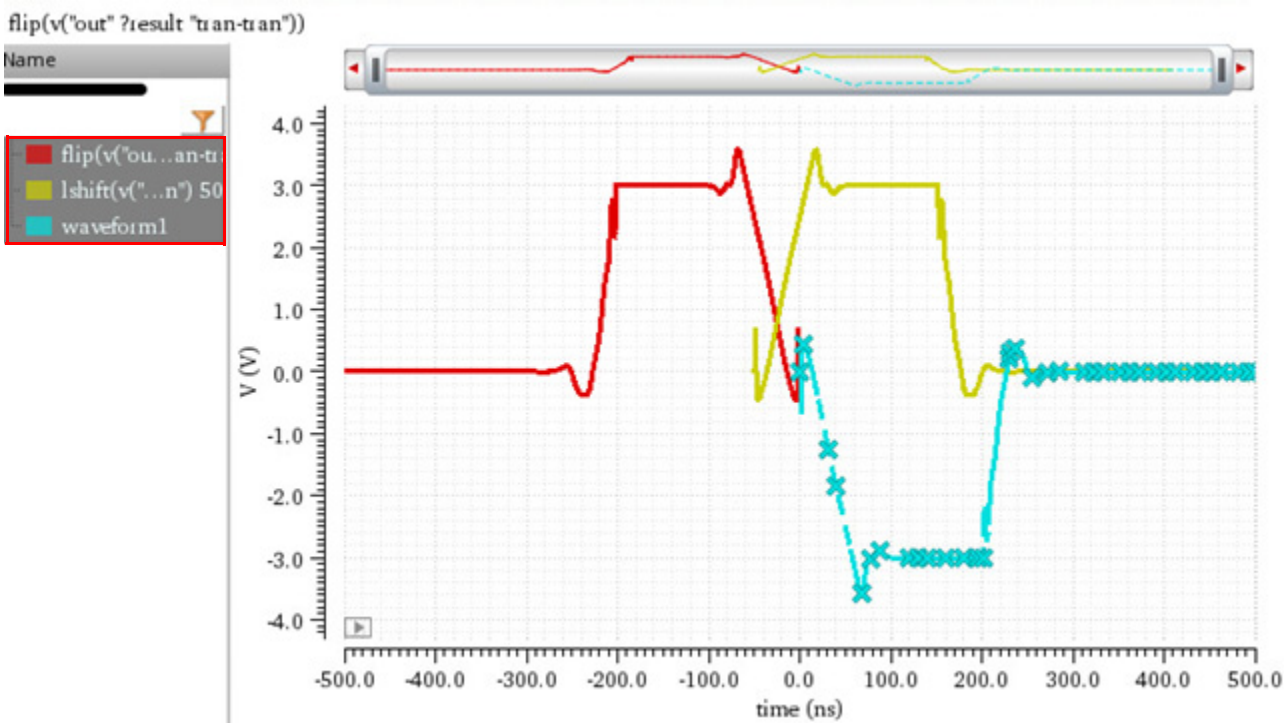
```
awvGetWaveNameList(awvGetCurrentWindow())
=>
((1 2)
    ("flip(v(\"out\" ?result \"tran-tran\"))" "lshift(v(\"out\" ?result \"tran-
tran\") 50ns )")
)
```

Note that waveforms are assigned with the lowest available index numbers `1` and `2`.

Now, plot an expression using `awvSimplePlotExpression`. Specify the name of the resulting waveform as `waveform1` using the `?expr` argument. Note that `waveform1` is plotted in the current Waveform window and existing waveforms are not overwritten because *`g_replace`* argument is set to `nil`.

```
awvSimplePlotExpression(awvGetCurrentWindow() "v(\"net10\" ?result \"tran-
tran\")-v(\"in_p\" ?result \"tran-tran\")" nil nil ?expr list("waveform1") ?color
list("y12") ?lineType list("line") ?lineStyle list("dash") ?lineThickness
list("fine") ?showSymbols list(t) ?dataSymbol list("X"))
```

```
=> t
```



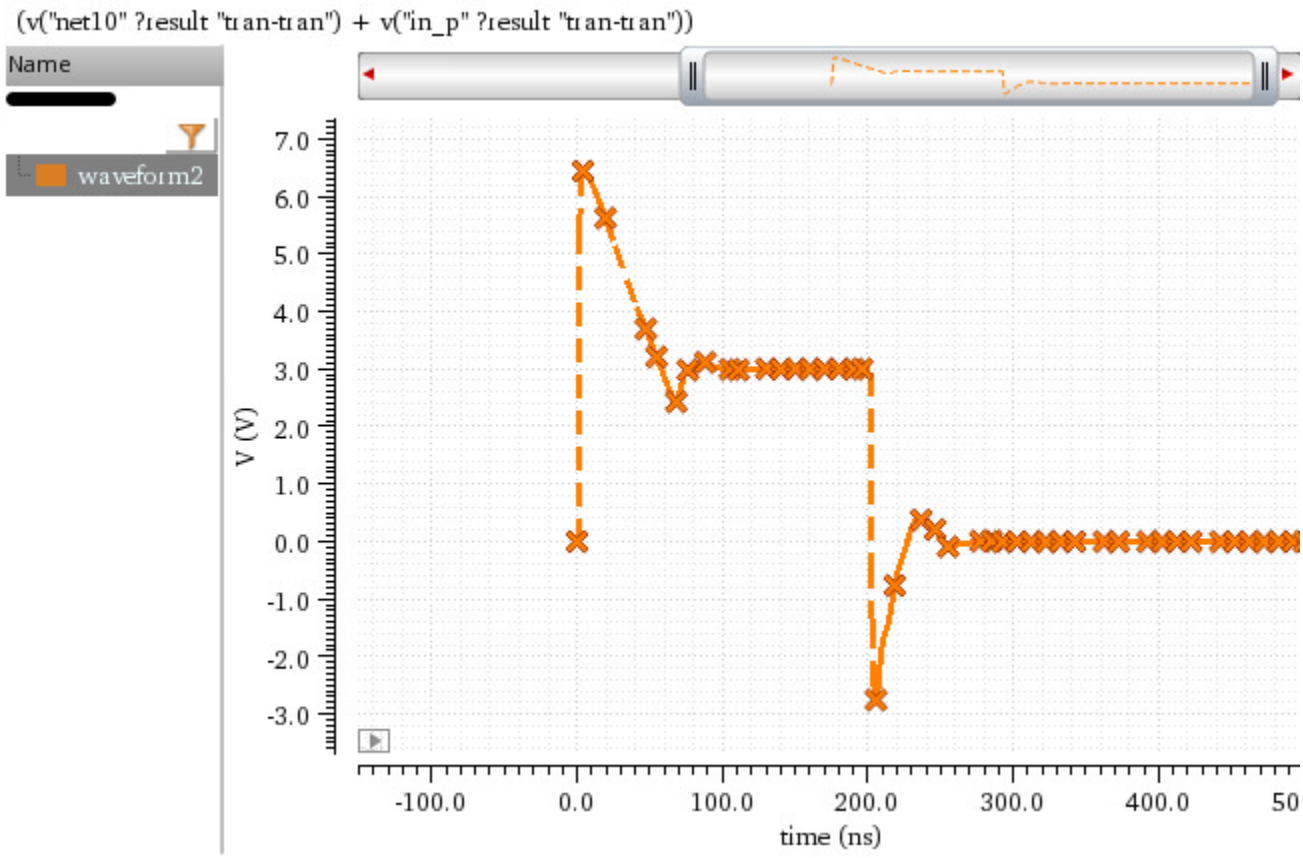
The following example returns the index numbers and names of the waveforms plotted in the current Waveform window. Note that `waveform1` is assigned with the next available index number, which is `3`.

```
awvGetWaveNameList(awvGetCurrentWindow())
=>
((1 2 3)
    ("flip(v(\"out\" ?result \"tran-tran\"))" "lshift(v(\"out\" ?result \"tran-
tran\") 50ns )" "waveform1")
)
```

Now, plot another expression using `awvSimplePlotExpression`. Specify the name of the resulting waveform as `waveform2` using the `?expr` argument. Note that `waveform2` is plotted in the current Waveform window and existing waveforms are overwritten because *`g_replace`* argument is set to `t`.

```
awvSimplePlotExpression(awvGetCurrentWindow() "v(\"net10\" ?result \"tran-
tran\")+v(\"in_p\" ?result \"tran-tran\")" nil t ?expr list("waveform2") ?color
list("y6") ?lineType list("line") ?lineStyle list("dash") ?lineThickness
list("fine") ?showSymbols list(t) ?dataSymbol list("X"))
```

```
=> t
```



The following example returns the index numbers and names of the waveforms plotted in the current Waveform window. Note that `waveform2` is assigned with the next available index number, which is `4`.

```
awvGetWaveNameList(awvGetCurrentWindow())
=>
((4)
    ("waveform2")
)
```

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`	Smith Axis menu is successfully displayed.
`nil`	Smith Axis menu cannot be displayed because of an error.

Examples

The following example shows how to run the `awvSmithAxisMenuCB` function.

```
awvSmithAxisMenuCB()
=> nil
```

awvTableSignals

```
awvTableSignals(
     l_signalList
     [ ?plotStyle t_plotStyle ]
     [ ?graphModifier t_graphModifier ]
     )
     => t / nil
```

Description

Displays the specified signals in the Results Display Window.

Arguments

l_signalList	List of signals to be displayed in the Results Display Window.
?plotStyle *t_plotStyle*	Plotting style.
?graphModifier *t_graphModifier*	Graph Modifier.

Value Returned

t	The specified signals are displayed in the Results Display Window.
nil	Signals cannot be displayed in the Results Display Window because of an error.

Examples

The following example displays the signal `out` in the Results Display Window from the specified results in a results directory.

```
awvTableSignals('(("/servers/user/design/ampsim.raw" (("tran-tran" ("out")
?plotStyle "Append" ?graphModifier "Magnitude")))))
=> t
```

awvUpdateAllWindows

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

Description

Updates the display of all Waveform windows.

Arguments

None

Value Returned

`t`	All Waveform windows are updated successfully.
`nil`	No Waveform windows are currently open.

Examples

The following example updates the display of all Waveform windows that are currently open.

```
awvUpdateAllWindows()
=> t
```

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	Subwindows of the specified Waveform window are updated successfully.
nil	The specified Waveform window does not exist.

awvZoomFit

```
awvZoomFit(
     w_windowID
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Fits the graph in the specified Waveform window.

Arguments

w_windowID	Waveform window ID.
?subwindow *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

t	Zoom fit operation is successful.
nil	The specified Waveform window or subwindow does not exist.

Examples

The following example returns the traces to their original size to fit in subwindow 4 of the specified Waveform window.

```
awvZoomFit(window(11) ?subwindow 4)
=> t
```

awvZoomGraphX

```
awvZoomGraphX(
     w_windowID
     l_minMax
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Zooms in or zooms out the graph according to the specified x-axis (independent axis) coordinates.

Arguments

w_windowID	Waveform window ID.

l_minMax	A list of two x-axis coordinates for performing zoom in or zoom out operation. The first value in the list is the minimum and the second is the maximum.
	The following notations are supported:
	`50n:220n`
	`list(50n 220n)`
	`list("50ns" "220ns")`
	`5e-8:2.2e-7`
	`list(5e-8 2.2e-7)`

`?subwindow` *x_subwindow*	Identification number of the subwindow, which is found in the top-right corner of the subwindow. If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Zoom in or zoom out operation is successful.
`nil`	The specified Waveform window or subwindow does not exist.

Examples

The following examples zoom in the x axis from `x=50ns` to `x=220n`s in the current subwindow of the current Waveform window.

```
awvZoomGraphX(awvGetCurrentWindow() 50n:220n ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
awvZoomGraphX(awvGetCurrentWindow() list(50n 220n) ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
awvZoomGraphX(awvGetCurrentWindow() list("50ns" "220ns") ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
awvZoomGraphX(awvGetCurrentWindow() 5e-8:2.2e-7 ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
awvZoomGraphX(awvGetCurrentWindow() list(5e-8 2.2e-7) ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
```

awvZoomGraphXY

```
awvZoomGraphXY(
     w_windowID
     l_xMinMax
     l_yMinMax
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Zooms in or zooms out the graph according to the specified y-axis (dependent axis) coordinates.

Arguments

w_windowID	Waveform window ID.

l_xMinMax	A list of two x-axis coordinates for performing zoom in or zoom out operation. The first value in the list is the minimum and the second is the maximum.
	The following notations are supported:
	`50n:220n`
	`list(50n 220n)`
	`list("50ns" "220ns")`
	`5e-8:2.2e-7`
	`list(5e-8 2.2e-7)`

l_yMinMax

A list of two y-axis coordinates for performing zoom in or zoom out operation. The first value in the list is the minimum value and the second value is the maximum value.

The following notations are supported:

```
750m:900m

list(750m 900m)

list("750mv" "900mv")

7.5e-1:9e-1

list(7.5e-1 9e-1)
```

?stripNumber *x_stripNumber*

Strip number to identify the strip on which zoom operation is to be performed.

If you do not specify this argument, the currently selected strip is used.

If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the zoom operation is performed on the trace with strip number `1`.

?subwindow *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Zoom in or zoom out operation is successful.
`nil`	The specified Waveform window, subwindow, or the strip does not exist.

Examples

The following example returns a list of strip numbers of the traces plotted in the current subwindow of the current Waveform window:

```
awvGetStripNumbersList(awvGetCurrentWindow() ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> (3 1 2)
```

The following example zooms in the x axis from `x=50ns` to `x=100ns` and the y axis from `y=750mV` to `y=900mV` of the trace with the strip number `2` in the current subwindow of the current Waveform window.

```
awvZoomGraphXY(awvGetCurrentWindow() list("50ns" "100ns") 7.5e-1:9e-1 ?stripNumber
2 ?subwindow awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
```

awvZoomGraphY

```
awvZoomGraphY(
     w_windowID
     l_minMax
     [ ?stripNumber x_stripNumber ]
     [ ?subwindow x_subwindow ]
     )
     => t / nil
```

Description

Zooms in or zooms out the graph according to the specified y-axis (dependent axis) coordinates.

Arguments

w_windowID Waveform window ID.

l_minMax A list of two y-axis coordinates for performing zoom in or zoom out operation. The first value in the list is the minimum value and the second value is the maximum value.

The following notations are supported:

```
750m:900m
```

```
list(750m 900m)
```

```
list("750mv" "900mv")
```

```
7.5e-1:9e-1
```

```
list(7.5e-1 9e-1)
```

?stripNumber *x_stripNumber*

Strip number to identify the strip on which zoom operation is to be performed. If you do not specify this argument, the currently selected strip is used.

If no strip is currently selected in the specified subwindow or the current subwindow, the default value `1` is used, which means that the operation is performed on the trace with strip number `1`.

?subwindow *x_subwindow*

Identification number of the subwindow, which is found in the top-right corner of the subwindow.

If you do not specify this argument, the current subwindow is used.

Value Returned

`t`	Zoom in or zoom out operation is successful.
`nil`	The specified Waveform window, subwindow, or the strip does not exist.

Examples

The following example returns a list of strip numbers of the traces plotted in the current subwindow of the current Waveform window:

```
awvGetStripNumbersList(awvGetCurrentWindow() ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> (3 1 2)
```

The following examples zoom in the y axis from `y=750mV` to `y=900mV` of the trace with the strip number `2` in the current subwindow of the current Waveform window.

```
awvZoomGraphY(awvGetCurrentWindow() 750m:900m ?stripNumber 2 ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
awvZoomGraphY(awvGetCurrentWindow() list(750m 900m) ?stripNumber 2 ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
awvZoomGraphY(awvGetCurrentWindow() list("750mV" "900mV") ?stripNumber 2
?subwindow awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
awvZoomGraphY(awvGetCurrentWindow() 7.5e-1:9e-1 ?stripNumber 2 ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
awvZoomGraphY(awvGetCurrentWindow() list(7.5e-1 9e-1) ?stripNumber 2 ?subwindow
awvGetCurrentSubwindow(awvGetCurrentWindow()))
=> t
```

famGetExpr

```
famGetExpr(
     o_waveform
     )
     => t_expr / nil
```

Description

Returns the expression of the specified waveform.

Arguments

o_waveform	ID of the waveform object whose expression is to be returned.

Value Returned

t_expr	Expression set to the specified waveform object.
nil	The expression of the specified waveform object cannot be returned because no expression has been set for the specified waveform.

Examples

The following example returns the expression set for the specified waveform object, waveform.

```
waveform=expr(x sin(x) linRg(0 20.0 .5))
; Creates a waveform object representing the waveform sin(x) from x=0.0 to x=20.0.
The expression is evaluated at x=0.0, 0.5, 1.0, 1.5, ..., 20.0.
=> srrWave:0x327010b0
famGetExpr(waveform)
; Returns the expression for the specified waveform object, waveform.
=>
expr(x
    sin(x)
    linRg(0 20.0 0.5)
)
```

famSetExpr

```
famSetExpr(
     o_waveform
     t_expr
     )
     => t_expr / nil
```

Description

Sets the specified expression for the waveform.

Arguments

o_waveform	ID of the waveform object for which the expression is to bet set.
t_expr	Expression to be set to the waveform.

Value Returned

t_expr	The expression set for the specified waveform object.
nil	The expression cannot be set for the specified waveform object.

Examples

The following example sets the specified expression for the waveform object, `waveform`.

```
waveform=expr(x sin(x) linRg(0 20.0 .5))
; Creates a waveform object representing the waveform sin(x) from x=0.0 to x=20.0.
The expression is evaluated at x=0.0, 0.5, 1.0, 1.5, ..., 20.0.
=> srrWave:0x327010b0

famGetExpr(waveform)
; Returns the expression for the specified waveform object, waveform.
=>
expr(x
    sin(x)
    linRg(0 20.0 0.5)
)
famSetExpr(waveform "sine_expression")
; Sets the specified expression "sine_expression" to the waveform object, waveform.
```

```
=> "sine_expression"
```

vvGetGraphBackground

```
vvGetGraphBackground(
     [ w_windowID ]
     )
     => t_backgroundColor / nil
```

Description

Returns the background color of the specified Waveform window or the current Waveform window.

Arguments

w_windowID	Waveform window ID. If you do not specify this argument, the current Waveform window is used.

Value Returned

t_backgroundColor	The background color of the specified Waveform window or the current waveform window.
nil	Either no Waveform window is currently open or the specified Waveform window does not exist.

Examples

The following examples return the background color of the current Waveform window.

```
vvGetGraphBackground(awvGetCurrentWindow())
=>
(("graphWindow[2.2.2]"
    (("background" "#000000"))
    )
)

vvGetGraphBackground()
=>
(("graphWindow[2.2.2]"
    (("background" "#000000"))
```

```
    )
)
```

The following example returns the background color of the specified Waveform window.

```
vvGetGraphBackground(window(3))
=>
(("graphWindow[1.1.1]"
    (("background" "#000000"))
    )
)
```

The hex code `#000000` indicates that the background color of the Waveform window is `black`.

vvSetGraphBackground

```
vvSetGraphBackground(
     t_backgroundColor
     [ w_windowID ]
     )
     => t / nil
```

Description

Sets the specified background color to the specified Waveform window or the current Waveform window.

Arguments

t_backgroundColor

Background color to be set to the Waveform window.

You can either specify the name of the color or the hex code of the color.

For example, `"yellow"` or `"#ffff00"`.

w_windowID

Waveform window ID.

If you do not specify this argument, the current Waveform window is used.

Value Returned

`t`

The specified background color is set to the Waveform window.

`nil`

Indicates one of the following:

- The specified background color is not a valid color.
- Either no Waveform window is currently open or the specified Waveform window does not exist.

Examples

The following examples set `yellow` as the background color of the current Waveform window.

```
vvSetGraphBackground("#ffff00" awvGetCurrentWindow())
```

```
vvSetGraphBackground("#ffff00")
vvSetGraphBackground("yellow" awvGetCurrentWindow())
vvSetGraphBackground("yellow")
```

You can also assign a bindkey to change the background color of the current Waveform window.

The following example assigns the bindkey `Ctrl + B` to change the background color of the current Waveform window to `yellow`.

```
hiSetBindKey("vivaGraph" "Ctrl<Key>B"
"vvSetGraphBackground(eval(vivaBackgroundColor))")
vivaBackgroundColor="yellow"
=> t
=> "yellow"
```

3

Results Browser Functions

The Results Browser SKILL functions let you perform various operations on Results Browser. These functions have the prefix `rdb`.

This topic lists Results Browser functions that are available in Virtuoso Visualization and Analysis XL.

rdbLoadResults

```
rdbLoadResults(
     t_sessionName
     t_resultsDir
     )
     => t / nil
```

Description

Loads the simulation results located at results directory into the Results Browser associated with the specified session.

Arguments

t_sessionName	Name of the session in which results are to be loaded in the Results Browser. Specify `"unbound"` if Virtuoso Visualization and Analysis XL is running in standalone mode and the Results Browser window is open.
t_resultsDir	Path to the simulation results.

Value Returned

`t`	Simulation results are loaded successfully.
`nil`	Simulation results cannot be loaded because of an error.

Examples

The following example returns the name of the ADE session associated with the current window.

```
axlSession=axlGetWindowSession()
=> "fnxSession0"
```

The following examples load simulation results of the specified results directory into the Results Browser associated with the session `fnxSession0`.

```
rdbLoadResults(axlSession "/home/user/LoopFinder/simulation/lib/cell/maestro/
results/maestro/ExplorerRun.0/1/test/psf")
=> t
```

```
rdbLoadResults("fnxSession0" "/home/user/LoopFinder/simulation/lib/cell/maestro/
results/maestro/ExplorerRun.0/1/test/psf")
=> t
```

The following example loads simulation results of the specified results directory into the Results Browser when Virtuoso Visualization and Analysis XL is running in standalone mode.

```
rdbLoadResults("unbound" "/home/user/LoopFinder/simulation/lib/cell/maestro/
results/maestro/ExplorerRun.0/1/test/psf")
=> t
```

rdbReloadResults

```
rdbReloadResults(
     t_sessionName
     t_resultsDir
     )
     => t / nil
```

Description

Reloads the simulation results located at results directory into the Results Browser associated with the specified session.

It is recommended to reload results during the successive simulation runs.

Arguments

t_sessionName	Name of the session in which results are to be reloaded in the Results Browser. Specify `"unbound"` if Virtuoso Visualization and Analysis XL is running in standalone mode and the Results Browser window is open.
t_resultsDir	Path to the simulation results.

Value Returned

`t`	Simulation results are reloaded successfully.
`nil`	Simulation results cannot be reloaded because of an error.

Examples

The following example returns the name of the ADE session associated with the current window.

```
axlSession=axlGetWindowSession()
=> "fnxSession0"
```

The following examples reload simulation results of the specified results directory into the Results Browser associated with the session `fnxSession0`.

```
rdbReloadResults(axlSession "/home/user/LoopFinder/simulation/lib/cell/maestro/
results/maestro/ExplorerRun.0/1/test/psf")
```

```
=> t

rdbReloadResults("fnxSession0" "/home/user/LoopFinder/simulation/lib/cell/
maestro/results/maestro/ExplorerRun.0/1/test/psf")

=> t
```

The following example reloads simulation results of the specified results directory into the Results Browser when Virtuoso Visualization and Analysis XL is open in standalone mode.

```
rdbReloadResults("unbound" "/home/user/LoopFinder/simulation/lib/cell/maestro/
results/maestro/ExplorerRun.0/1/test/psf")

=> t
```

rdbSetCurrentDirectory

```
rdbSetCurrentDirectory(
     t_sessionName
     t_pathToDirectory
     )
     => t / nil
```

Description

Navigates to the specified directory in the Results Browser associated with the specified session.

Arguments

t_sessionName	Name of the session. Specify `"unbound"` if Virtuoso Visualization and Analysis XL is running in standalone mode and the Results Browser window is open.
t_pathToDirectory	Path to the directory inside the simulation results.

Value Returned

`t`	Specified directory is set as the current directory.
`nil`	Specified directory cannot be set as the current directory because of an error.

Examples

The following example returns the name of the ADE session associated with the current window.

```
axlSession=axlGetWindowSession()
=> "fnxSession0"
```

The following examples navigate to the `lf` directory inside the simulation results of the results directory in the Results Browser associated with the specified session.

```
rdbSetCurrentDirectory(axlSession "/home/user/LoopFinder/simulation/lib/cell/
maestro/results/maestro/ExplorerRun.0/1/test/psf/lf")
```

```
=> t

rdbSetCurrentDirectory("fnxSession0" "/home/user/LoopFinder/simulation/lib/cell/
maestro/results/maestro/ExplorerRun.0/1/test/psf/lf")

=> t
```

The following example navigates to the `lf` directory inside the simulation results stored in the specified directory when Virtuoso Visualization and Analysis XL is open in standalone mode.

```
rdbSetCurrentDirectory("unbound" "/home/user/LoopFinder/simulation/lib/cell/
maestro/results/maestro/ExplorerRun.0/1/test/psf/lf")

=> t
```

rdbShowDialog

```
rdbShowDialog(
     t_sessionName
     t_componentName
     t_dialogName
     t_actionName
     [ l_options ]
     )
     => t / nil
```

Description

Displays or hides the specified form associated with the Results Browser.

The specified form is displayed only when the Results Browser is already open.

Arguments

t_sessionName	Name of the session.
t_componentName	Name of the component. The value of this argument is always `"browser"`.
t_dialogName	Name of the form associated with the Results Browser. Valid values are as follows: ■ `findResults`: Displays or hides the Select Waveform Database form. ■ `saveAs`: Displays or hides the Export Waveforms form. ■ `sweepData`: Displays or hides the Data Browser: Set Sweep Ranges form.
t_actionName	Name of the action to be performed. Valid values are as follows: ■ `show`: Displays the specified form. ■ `hide`: Hides the specified form.

l_options	List of optional arguments that can be passed to the function. This argument is relevant only when the argument *t_dialogName* is set to `findResults`. For example: ■ `list(list("pathList" "/servers/user/testCase/design/ampsim.raw"))` ■ `'(("pathList" "/servers/user/testCase/design/ampsim.raw"))`

Value Returned

`t`	The specified form is displayed or hidden.
`nil`	The specified form cannot be displayed or hidden because of an error.

Examples

The following example opens the Select Waveform Database form.

```
rdbShowDialog("fnxSession0" "browser" "findResults" "show")
```

The following examples open the Select Waveform Database form and set the initial location to the specified directory.

```
rdbShowDialog("fnxSession0" "browser" "findResults" "show" list(list("pathList" "/servers/user/testCase/design/ampsim.raw")))
=> t
rdbShowDialog("fnxSession0" "browser" "findResults" "show" '(("pathList" "/servers/user/testCase/design/ampsim.raw") ))
=> t
```

The following example hides the Select Waveform Database form.

```
rdbShowDialog("fnxSession0" "browser" "findResults" "hide")
```

The following example opens the Export Waveforms form.

```
rdbShowDialog("fnxSession0" "browser" "saveAs" "show")
```

The following example hides the Export Waveforms form.

```
rdbShowDialog("fnxSession0" "browser" "saveAs" "hide")
```

The following example opens the Data Browser: Set Sweep Ranges form.

```
rdbShowDialog("fnxSession0" "browser" "sweepData" "show")
```

The following example hides the Data Browser: Set Sweep Ranges form.

```
rdbShowDialog("fnxSession0" "browser" "sweepData" "hide")
```

rdbUnloadResults

```
rdbUnloadResults(
     t_sessionName
     t_resultsDir
     )
     => t / nil
```

Description

Unloads the simulation results located at results directory from the Results Browser associated with the specified session.

It is recommended to unload results during the successive simulation runs to reduce resource consumption and remove clutter from the Results Browser.

Arguments

t_sessionName	Name of the session from which results are to be unloaded in the Results Browser. Specify `"unbound"` if Virtuoso Visualization and Analysis XL is running in standalone mode and the Results Browser window is open.
t_resultsDir	Path to the simulation results.

Value Returned

`t`	Simulation results are successfully unloaded.
`nil`	Simulation results cannot be unloaded because of an error.

Examples

The following example unloads the simulation results of the specified results directory from the Results Browser associated with the session `fnxSession0`.

```
rdbUnloadResults("fnxSession0" "./simulation/opamp090/full_diff_opamp/maestro/
results/maestro/Interactive.13/psf/TRAN/psf")
```

The following example unloads simulation results of the specified results directory from the Results Browser when Virtuoso Visualization and Analysis XL is open in standalone mode.

```
rdbUnloadResults("unbound" "/home/user/LoopFinder/simulation/lib/cell/maestro/
results/maestro/ExplorerRun.0/1/test/psf")
=> t
```

rdbWriteToFormat

```
rdbWriteToFormat(
     t_sessionName
     t_path
     t_format
     [ l_signals ]
     )
     => t / nil
```

Description

Exports the specified signals from the Results Browser associated with the specified session.

Arguments

t_sessionName	Name of the session.
t_path	Absolute or relative path to the file in which signals are to be exported.
t_format	Format of the file. Valid values are as follows: ■ CSV ■ VCSV ■ PSF ■ SST2 ■ Matlab ■ Spectre
l_signals	List of signals to be exported. If you do not specify a signal name in this argument, all the signals that are selected in the Results Browser are exported.

Value Returned

t	Signals are successfully exported.
nil	Signals cannot be exported because of an error.

Examples

The following example returns the ADE session name associated with the current window.

```
axlSession=axlGetWindowSession()
=> "fnxSession0"
```

The following example opens the Results Browser and returns ID of the Results Browser window.

```
vvDisplayBrowser()
=> dwindow:15
```

The following example loads simulation results of the specified results directory to the Results Browser associated with the specified session.

```
rdbLoadResults("fnxSession0" "./simulation/lib/cell/maestro/results/maestro/
Interactive.15/psf/TRAN/psf")
=> t
```

The following example navigates to the result `tran-tran` in the Results Browser window associated with the specified session.

```
rdbSetCurrentDirectory("fnxSession0" "./simulation/lib/cell/maestro/results/
maestro/Interactive.15/psf/TRAN/psf/tran-tran")
=> t
```

The following examples export signals `OUTN` and `OUTP` to the file `myExport1.vcsv` and `myExport2.vcsv`, respectively.

```
rdbWriteToFormat("fnxSession0" "/home/user/myExport1.vcsv" "VCSV" list(list("./
simulation/lib/cell/maestro/results/maestro/Interactive.15/psf/TRAN/psf"
list(list("tran-tran" list("OUTN" "OUTP"))))))
=> Exported to File /home/user/myExport1.vcsv
=> t


rdbWriteToFormat("fnxSession0" "/home/user/myExport2.vcsv" "VCSV" '(("./
simulation/lib/cell/maestro/results/maestro/Interactive.15/psf/TRAN/psf" (("tran-
tran" list("OUTN" "OUTP"))))))
=> Exported to File /home/user/myExport2.vcsv
=> t
```

Related Topics

rdbLoadResults

rdbSetCurrentDirectory

vvDisplayBrowser

vivaInitBindkeys

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

Description

Initializes the Virtuoso Visualization and Analysis XL bindkeys called from `viva.ini` context initialization file.

Arguments

None

Value Returned

`t`	Bindkeys are initialized successfully.
`nil`	Bindkeys cannot be initialized because of an error.

Examples

```
vivaInitBindkeys()
=> t
```

vivaIsVivaExecutable

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

Description

Checks whether the current application (binary) is Virtuoso Visualization and Analysis XL.

If you are running Virtuoso Visualization and Analysis XL in standalone mode (executable mode), you can use this function to avoid loading customizations that are not necessary for the application. This helps in improving startup time, memory usage, and performance of the application.

Arguments

None

Value Returned

`t`	Indicates that the current application is Virtuoso Visualization and Analysis XL.
`nil`	Indicates that the current application is Virtuoso.

Examples

The following example indicates that `layoutCustomizations.il` is loaded when Virtuoso Visualization and Analysis is not running in standalone mode (executable mode).

```
unless(vivaIsVivaExecutable()
        load("./layoutCustomizations.il")
        )
=> t
```

The following example indicates that the current executable application is Virtuoso Visualization and Analysis XL, and therefore, two custom functions `myFunction1` and `myFunction2` are added to the *Custom Functions* drop-down list in the *Function Panel* of Calculator.

The SKILL definitions and UI templates for both custom functions are saved in a single SKILL file `customCalFunction.il`.

```
if(vivaIsVivaExecutable()
    awvLoadCustomCalcFunction(
                                ?funcList list("myFunction1" "myFunction2")
                                ?fileName "/home/user/customCalFunction.il"
                              )
    )
=> t
```

vvDisplayBrowser

```
vvDisplayBrowser(
     )
     => l_windowID / nil
```

Description

Invokes the Results Browser within a window.

Arguments

None

Value Returned

l_windowID	Window ID of the Results Browser window.
nil	Results Browser cannot be invoked because of an error.

Examples

The following example invokes the Results Browser and returns the window ID.

```
vvDisplayBrowser()
=> dwindow:15
```

4

Calculator Functions

The Calculator SKILL functions let you perform various operations on Calculator.

You can also perform various calculations on the results of transient and ac analysis and plot outputs using Calculator functions.

This topic lists Calculator functions that are available in Virtuoso Visualization and Analysis XL.

caliRestoreDefaultWindowSize

calSetBuffer

dBm50ohm

dBm50ohmAny

expr

eyeBERLeft

eyeBERLeftApprox

eyeBERRight

eyeBERRightApprox

eyeHeightAtXY

eyeMask

eyeMaskViolationPeriodCount

eyePeakToPeakJitter

eyeWidthAtXY

famEval

firstVal

kurtosis

lastVal

leafValue

mu

Mu

mu_prime

Mu_prime

normalQQPValue

numConv

OS

OT

pvifreq

pvrfreq

skewness

swapSweep

topBaseLine

topLine

triggeredDelay

valueAt

vvDisplayCalculator

waveVsWave

aaSP

```
aaSP(
     n_portOrder1
     n_portOrder2
     [ t_dataDirectory ]
     )
     => o_waveform / nil
```

Description

Returns the S-Parameter waveform for a multi-port network for the specified port numbers. The S-Parameters describe the response of an N-port network to voltage signals at each port.

Arguments

n_portOrder1	Port number of the responding port.
n_portOrder2	Port number of the incident port.
t_dataDirectory	Results directory containing results of S-Parameter analysis. This is an optional argument.

Value Returned

o_waveform	S-Parameter waveform.
nil	S-Parameter waveform cannot be returned because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results of S-Parameter analysis stored in the specified directory.

```
openResults("/servers/user/spAnalysis/simulation/ampTest/spectre/schematic/psf")
=> "/servers/user/spAnalysis/simulation/ampTest/spectre/schematic/psf"
```

The following example returns a list of available results in the currently open results directory.

```
results()
=>
(dcOp dcOpInfo ac sp model
    instance output designParamVals primitives subckts
    variables
)
```

The following example selects the `sp` results. These are the results of S-Parameter analysis for a two-port network.

```
selectResults('sp)
=> stdobj@0x31bd4b60
```

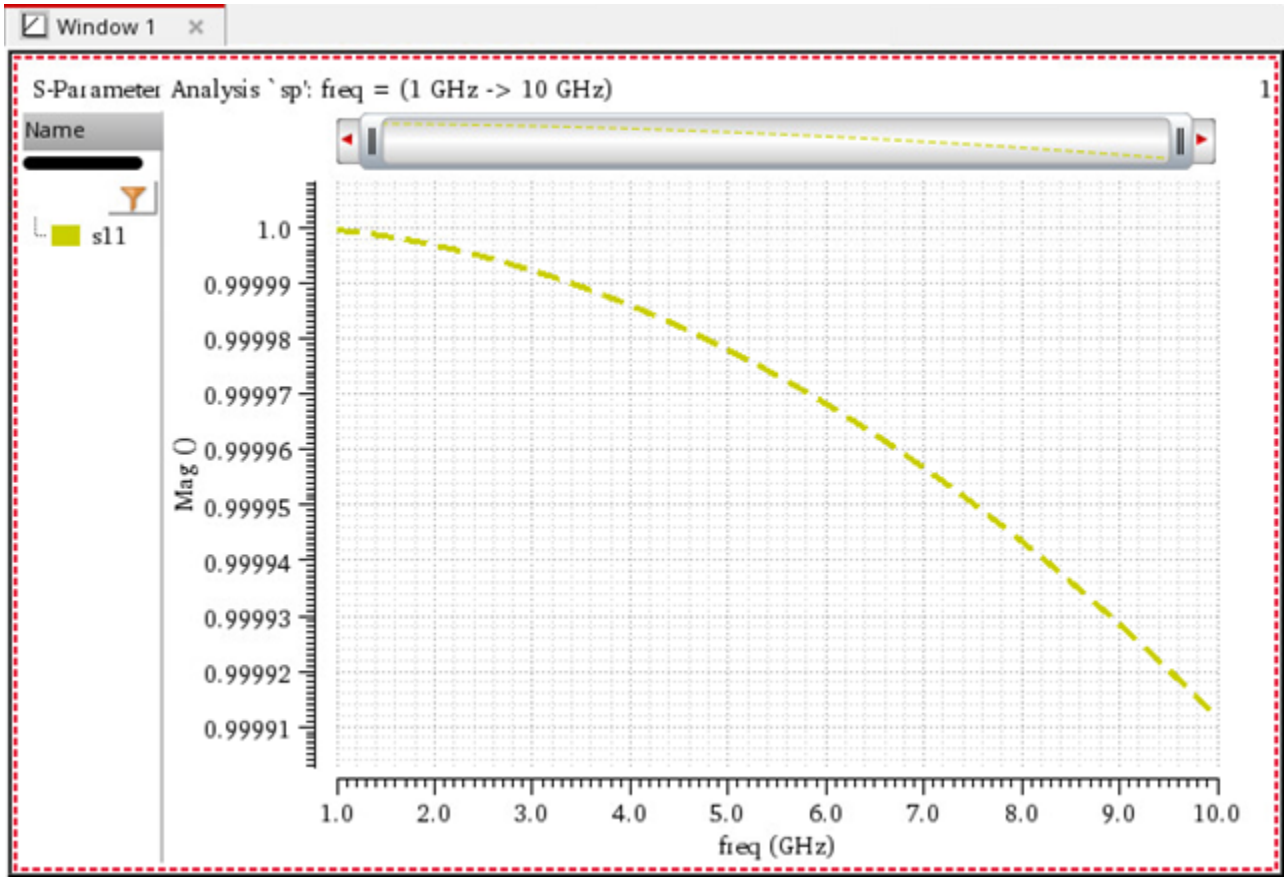
The following example returns a waveform object `s11`, which represents the response at port `1` due to a signal at port `1`.

```
s11=aaSP(1 1)
=> srrWave:0x35e7a020
```

The following example plots the waveform `s11` in the Waveform window having window ID `3`.

```
awvPlotWaveform(
                window(3)
                list(s11)
                ?expr list("s11")
                ?color list("y8")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                )
```

```
=> t
```



The following example creates another Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:4
```

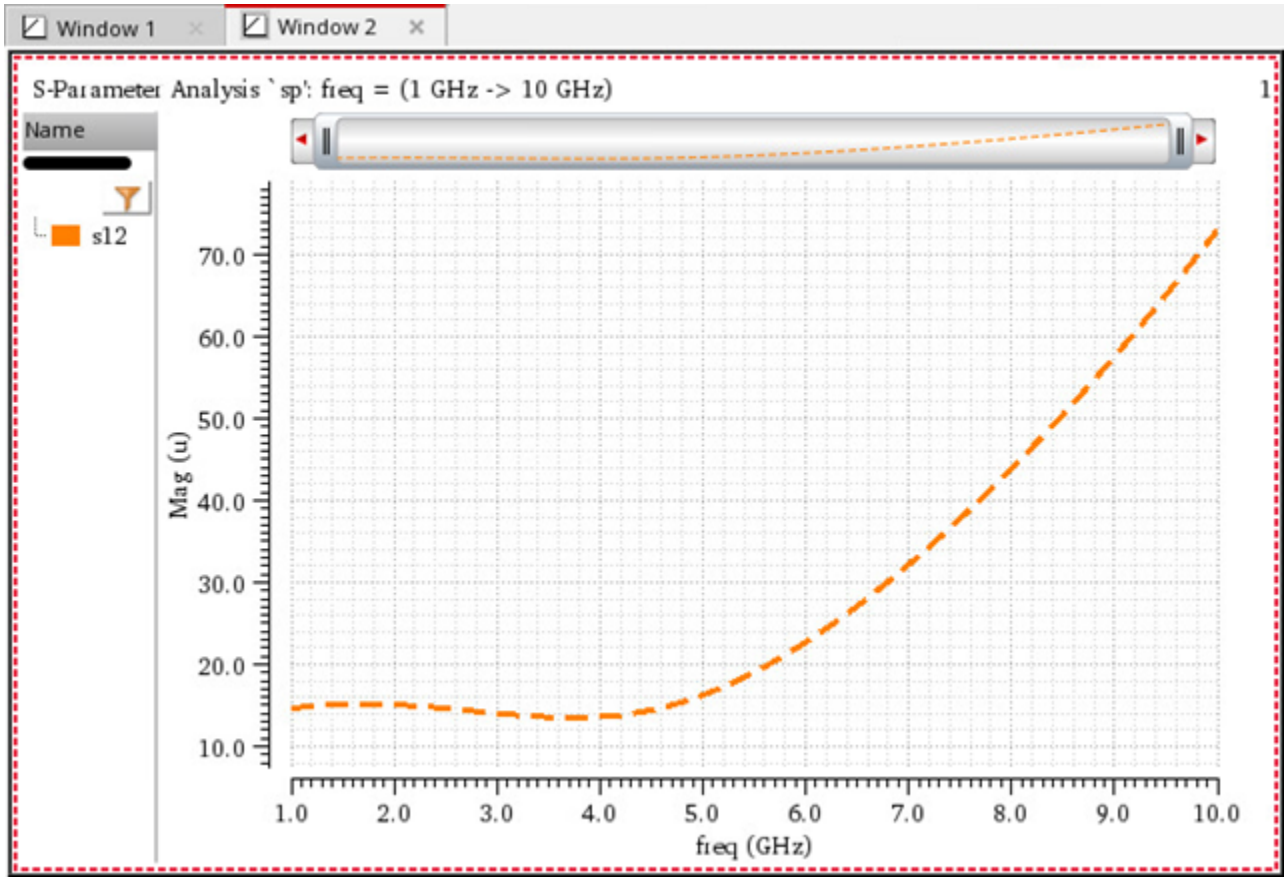
The following example returns a waveform object `s12`, which represents the response at port `1` due to a signal at port `2`.

```
s12=aaSP(1 2)
=> srrWave:0x35e7a030
```

The following example plots the waveform `s12` in the Waveform window having window ID `4`.

```
awvPlotWaveform(
                window(4)
                list(s12)
                ?expr list("s12")
                ?color list("y6")
                ?index list(1)
```

```
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                )
=> t
```



The following example creates another Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:5
```

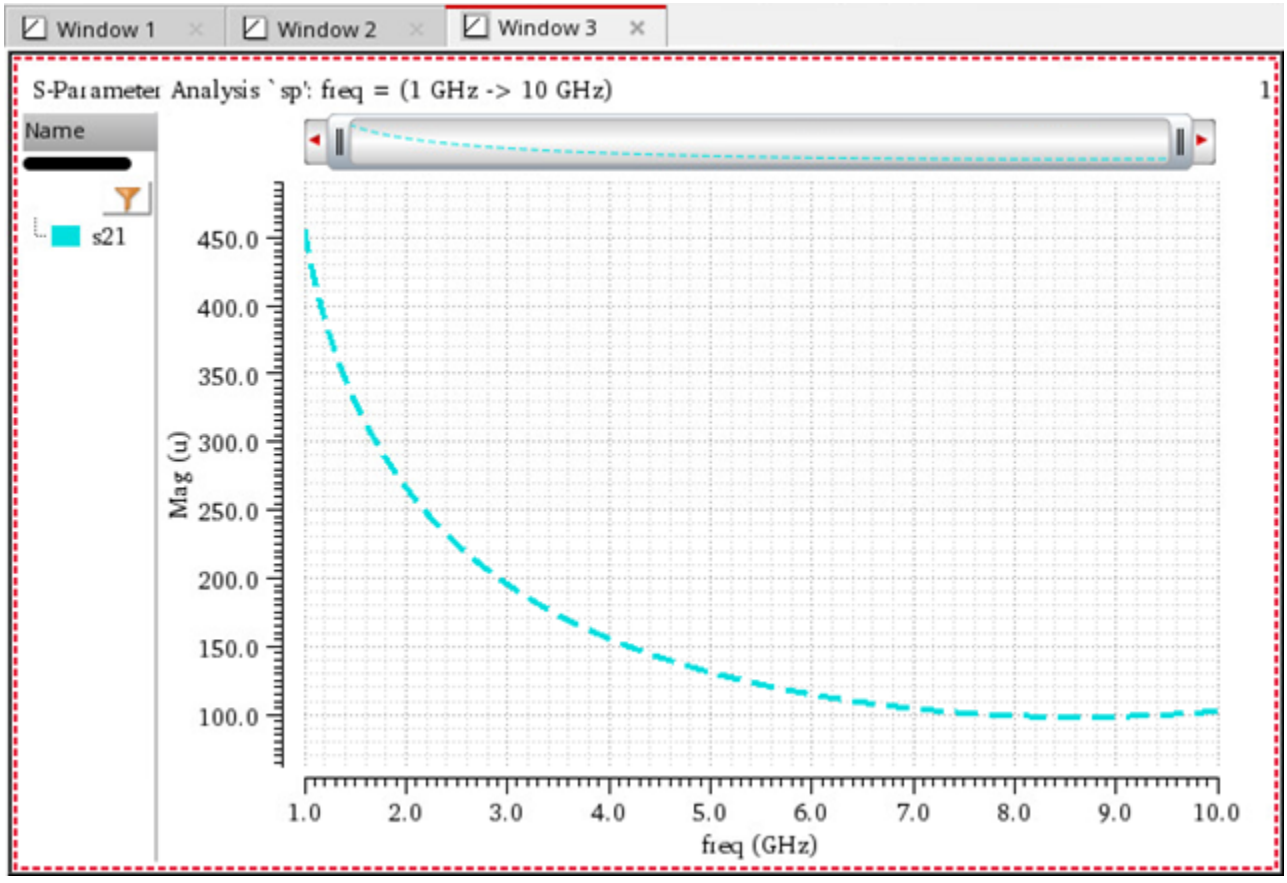
The following example returns a waveform object `s21`, which represents the response at port `2` due to a signal at port `1`.

```
s21=aaSP(2 1)
=> srrWave:0x35e7a040
```

The following example plots the waveform `s21` in the Waveform window having window ID `5`.

```
awvPlotWaveform(
                window(5)
```

```
                list(s21)
                ?expr list("s21")
                ?color list("y12")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                )
=> t
```



The following example creates another Waveform window and returns its window ID.

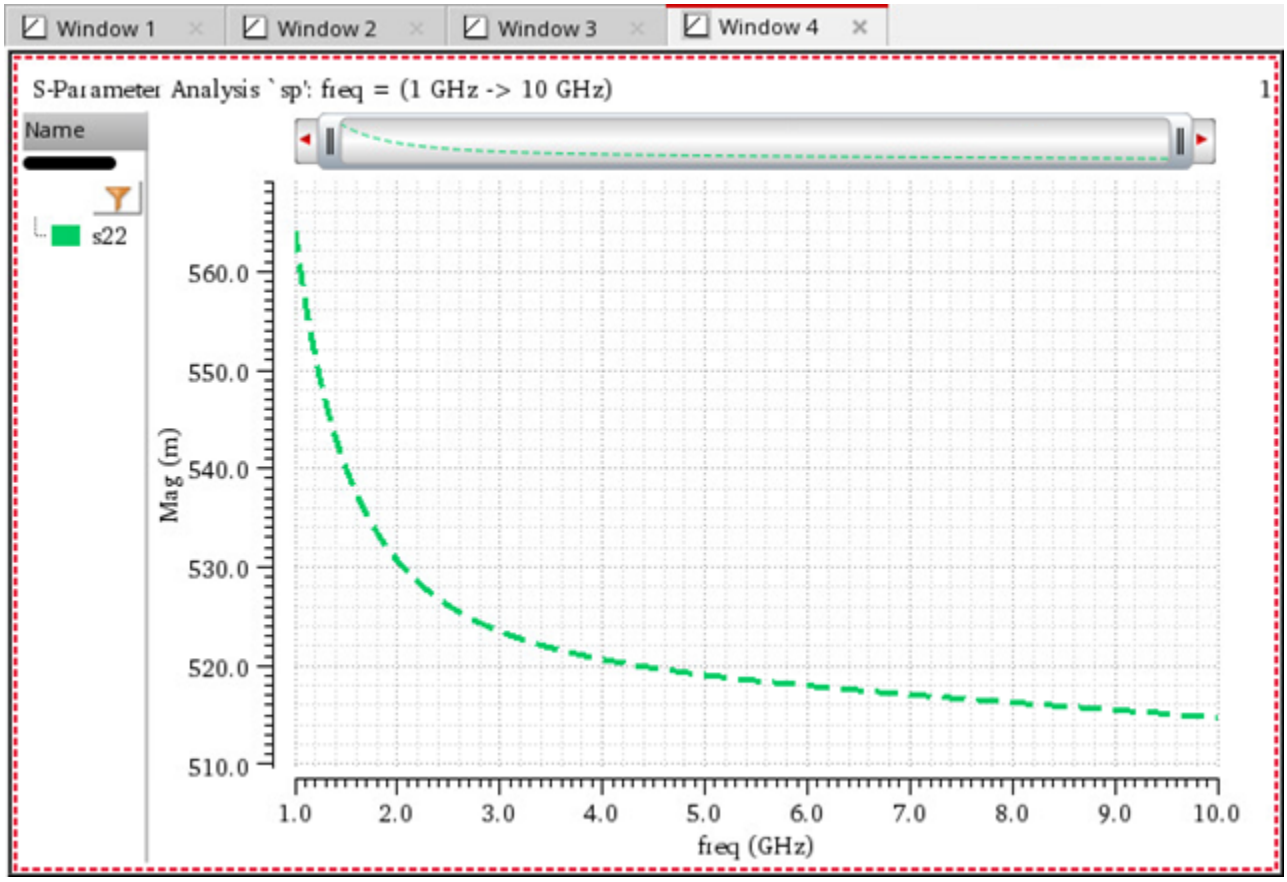
```
awvCreatePlotWindow()
=> window:6
```

The following example returns a waveform object `s22`, which represents the response at port `2` due to a signal at port `2`.

```
s22=aaSP(2 2)
=> srrWave:0x35e7a050
```

The following example plots the waveform `s22` in the Waveform window having window ID `6`.

```
awvPlotWaveform(
                window(6)
                list(s22)
                ?expr list("s22")
                ?color list("y18")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                )
=> t
```



The following examples create waveform objects `s11`, `s12`, `s21`, and `s22`. Note that the directory `/servers/user/spAnalysis/simulation/ampTest/spectre/schematic/psf` contains simulation results of S-Parameter analysis for a two-port network.

```
s11=aaSP(1 1 "/servers/user/spAnalysis/simulation/ampTest/spectre/schematic/psf")
=> srrWave:0x3559d020
```

```
s12=aaSP(1 2 "/servers/user/spAnalysis/simulation/ampTest/spectre/schematic/psf")
=> srrWave:0x3559d030
s21=aaSP(2 1 "/servers/user/spAnalysis/simulation/ampTest/spectre/schematic/psf")
=> srrWave:0x3559d040
s22=aaSP(2 2 "/servers/user/spAnalysis/simulation/ampTest/spectre/schematic/psf")
=> srrWave:0x3559d050
```

adtFFT

```
adtFFT(
     l_list
     )
     => l_result / nil
```

Description

Calculates the fast Fourier transform (FFT) of the specified list.

Arguments

l_list	List of values for which you want to generate FFT.

Value Returned

l_result	List containing FFT results.
nil	FFT cannot be calculated because of an error.

Examples

The following example calculates FFT for the specified values.

```
adtFFT(list(3 9 2 5 1))
=> (complex(20, 0) complex(0.427051, -5.8451) complex(-2.92705, -7.55545)
complex(-2.92705, 7.55545) complex(0.427051, 5.8451))
```

adtIFFT

```
adtIFFT(
     l_list
     )
     => l_result / nil
```

Description

Calculates the inverse discrete Fourier transform of the specified list.

Arguments

l_list	List of values for which you want to generate the inverse discrete Fourier transform.

Value Returned

l_result	List containing inverse discrete Fourier transform results.
nil	Inverse discrete Fourier transform cannot be calculated because of an error.

Examples

The following examples calculate inverse discrete Fourier transform of the specified values.

```
adtIFFT(list(complex(20, 0) complex(0.427051, -5.8451) complex(-2.92705, -7.55545)
complex(-2.92705, 7.55545) complex(0.427051, 5.8451)))
=> (complex(3, 0) complex(9, 0) complex(2, 0) complex(5, 0) complex(0.999999, 0))
adtIFFT(adtFFT(list(3 9 2 5 1)))
=> (complex(3, 0) complex(9, 0) complex(2, 0) complex(5, 0) complex(1, 0))
```

appendWaves

```
appendWaves(
     o_waveform1
     o_waveform2
     [ o_waveformN ]
     )
     => o_appendedWaveform / nil
```

Description

Appends a series of input waveforms in x-vector direction into a single output waveform.

Ensure that x vectors of input waveforms are in sequence.

Arguments

o_waveform1	The first waveform
o_waveform2	The second waveform.
o_waveformN	The n^{th} waveform. This is an optional argument.

Value Returned

o_appendedWaveform	The output waveform is created after appending the specified waveforms.
nil	The specified waveforms cannot be appended because of an error.

Examples

The following examples create three waveforms objects w1, w2, and w3, representing waveforms created by plotting specified x and y-vector values.

```
w1=drCreateWaveform(drCreateVec('double list(1 2 3 4 5)) drCreateVec('double
list(10 15 20 25 30)))
=> srrWave:0x2df1c020
w2=drCreateWaveform(drCreateVec('double list(6 7 8 9 10)) drCreateVec('double
list(35 40 45 50 55)))
=> srrWave:0x2df1c030
```

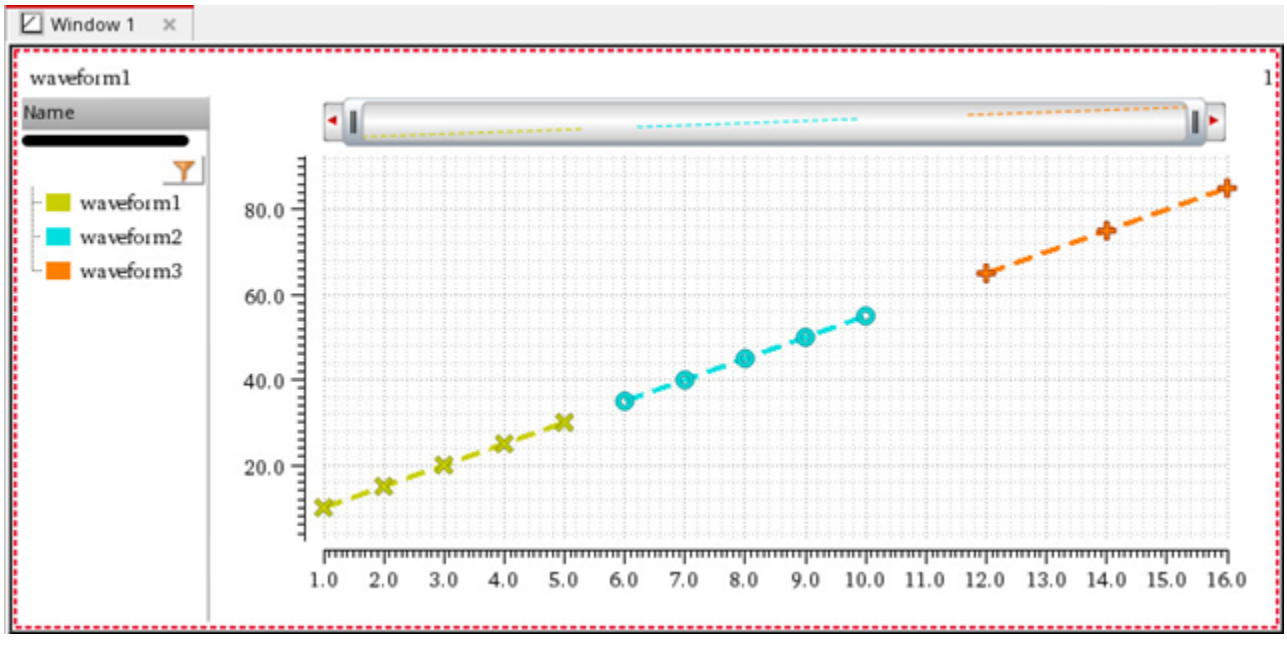
```
w3=drCreateWaveform(drCreateVec('double list(12 14 16)) drCreateVec('double
list(65 75 85)))
=> srrWave:0x2df1c040
```

The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example plots waveforms `w1`, `w2`, and `w3` in the Waveform window you created using the function `awvCreatePlotWindow`.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(w1 w2 w3)
                ?expr list("waveform1" "waveform2" "waveform3")
                ?color list("y8" "y12" "y6")
                ?index list(1 2 3)
                ?lineType list("line" "line" "line")
                ?lineStyle list("dash" "dash" "dash")
                ?lineThickness list("thick" "thick" "thick")
                ?showSymbols list(t t t)
                ?dataSymbol list("x" "O" "+")
                )
=> t
```



The following example appends input waveforms w1, w2, and w3 and creates an output waveform `w4`.

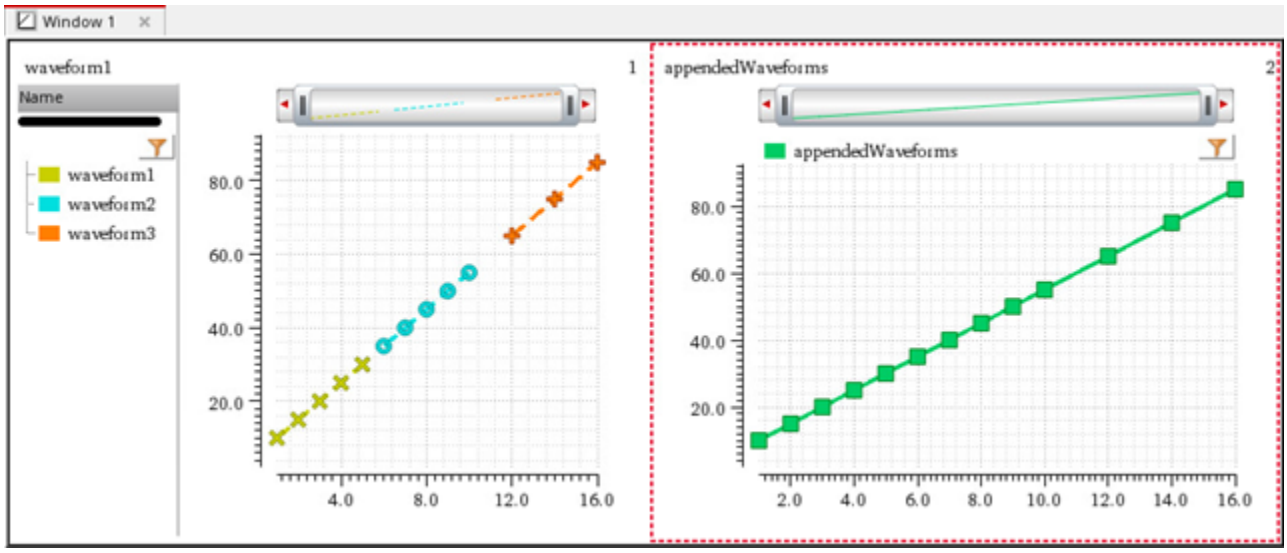
```
w4=appendWaves(w1 w2 w3)
=> srrWave:0x2df1c060
```

The following example adds a subwindow to the Waveform window. The newly created subwindow `2` is also selected as the current subwindow.

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

The following example plots the waveform `w4`, which is created by appending input waveforms `w1`, `w2`, and `w3` in the direction of their `x` vectors. The appended waveform `w4` is plotted in the current subwindow of the current Waveform window.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(w4)
                ?expr list("appendedWaveforms")
                ?color list("y18")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list(3)
                )
=> t
```



armSetCalc

```
armSetCalc(
     s_name
     g_value
     )
     => g_value
```

Description

Sets the specified property of the Calculator resource to the specified value.

Arguments

s_name	Name of the Calculator resource.
g_value	Value to be set.

Value Returned

g_value	Value of the property of the Calculator resource.

Examples

The following example sets the number of stacks to be displayed in the Calculator to `10`.

```
armSetCalc('numStack 10)
=> 10
```

baseLine

```
baseLine(
     o_waveform
     )
     => n_value / nil
```

Description

Returns the baseline value of the specified transient waveform.

Arguments

o_waveform	Waveform whose baseline value is to be calculated.

Value Returned

n_value	Baseline value of the specified transient waveform.
nil	Baseline value cannot be calculated because of an error.

Examples

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

The `tran-tran` result of the results directory `ampsim.raw` contain a transient signal `out`.

```
waveform=v("out" ?result "tran-tran")
=> srrWave:0x34930020
```

The following example calculates the baseline value of the transient signal out.

```
baseLine(waveform)
=> -0.03882059
```

busTransition

```
busTransition(
     o_busWaveform
     t_yFrom
     t_yTo
     [ n_nth ]
     [ t_xName ]
     )
     => t_nthTransition / t_nthLastTransition / t_allTransitions / nil
```

Description

Returns the time when a bus value is changed from a specified value to another specified value.

Arguments

o_busWaveform	Waveform object representing a bus.
t_yFrom	The starting bus value from which the bus is changed..
t_yTo	The target bus value to which the bus is changed.
n_nth	Specifies the n^{th} transition. ■ n>0: The function returns x-axis value at which the n^{th} time bus transition from *t_yFrom* to *t_yTo* occurs. ■ n<0: The function returns x-axis value at which the n^{th} last time bus transition from *t_yFrom* to *t_yTo* occurs. ■ n=0: The function returns all x-axis values at which bus transitions from *t_yFrom* to *t_yTo* occur. This is an optional argument. If you do not specify this argument, default value n=0 is used.
t_xName	Specifies the x-axis units of the generated waveform when n=0. Valid values are time and cycle. Default value is time. This is an optional argument.

Value Returned

t_nthTransition	The x-axis value at which the nth time bus transition from *t_yFrom* to *t_yTo* occurs.
t_nthLastTransition	The x-axis value at which the nth last time bus transition from *t_yFrom* to *t_yTo* occurs.
t_allTransitions	All x-axis values at which bus transitions from *t_yFrom* to *t_yTo* occur.
nil	Either the specified bus does not exist or the specified transition never occurs in the bus.

Examples

The following example creates an empty Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

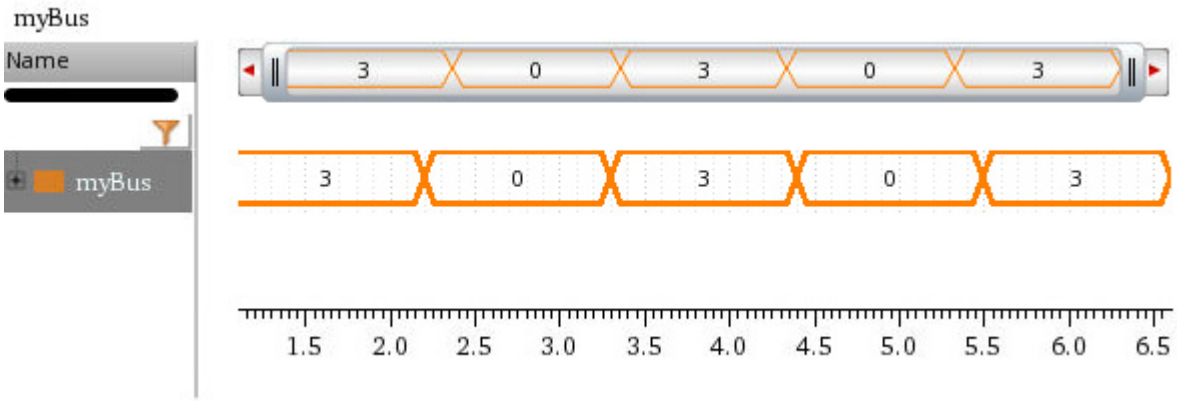
The following example creates a bus with x- and y-vector values.

```
xvec=drCreateVec('double '(1.1 2.2 3.3 4.4 5.5 6.6))
=> srrVec:0x31403020
yvec=drCreateVec('busVec '("11" "00" "11" "00" "11" "00"))
=> srrVec:0x31403030
myBus=drCreateWaveform(xvec yvec)
=> srrWave:0x31451020
```

The following example plots the waveform represented by waveform object myBus.

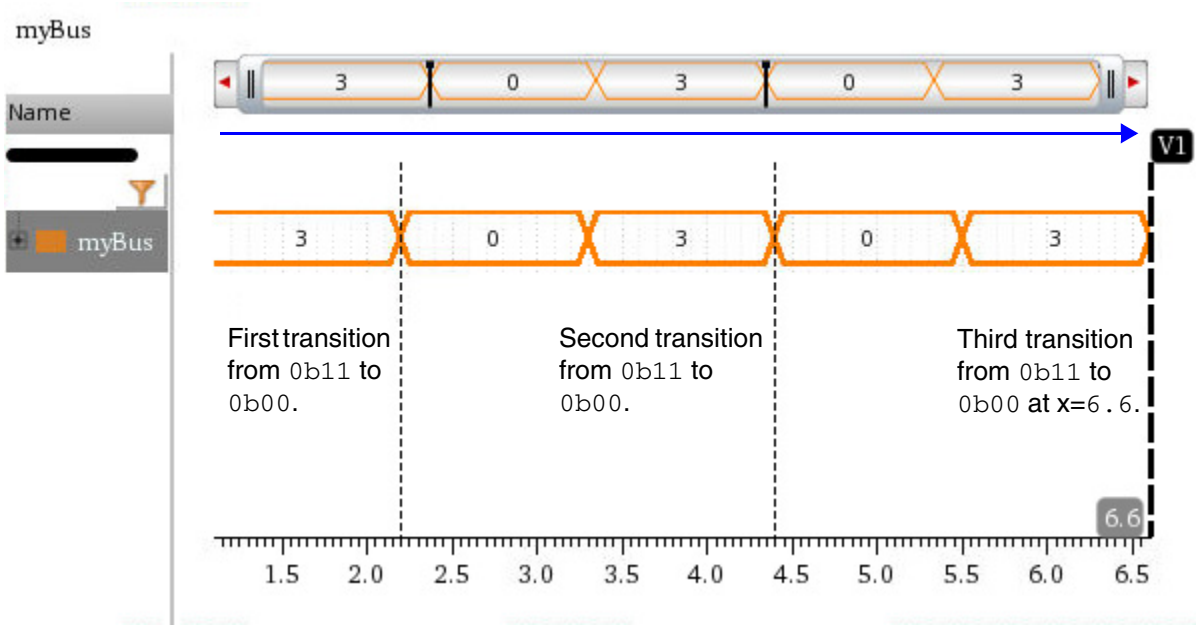
```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(myBus)
                ?expr list("myBus")
                ?color list("y6")
                )
```

```
=> t
```



The following example returns the x-axis value at which the bus `myBus` transitions from binary value `0b11` ($3_{decimal}$) to binary value `0b00` ($0_{decimal}$) the **third** time.

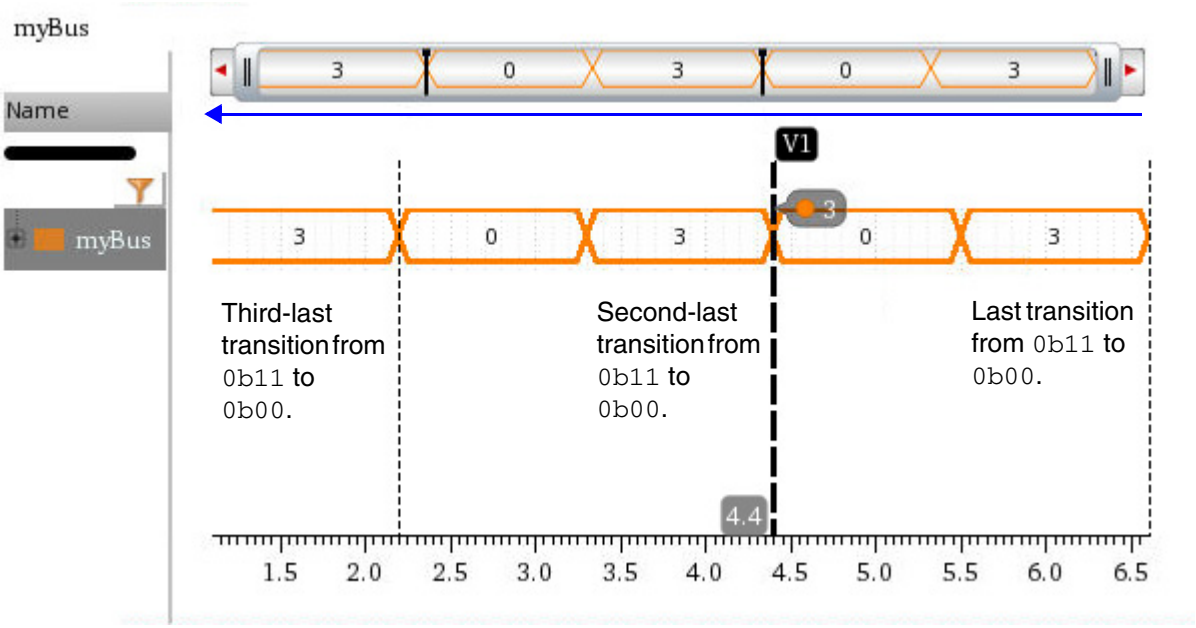
```
busTransition(myBus "0b11" "0b00" 3)
=> 6.6
```



The following example returns the x-axis value at which the bus `myBus` transitions from binary value `0b11` ($3_{decimal}$) to binary value `0b00` ($0_{decimal}$) the **second-last** time.

```
busTransition(myBus "0b11" "0b00" -2)
```

```
=> 4.4
```



The following example returns all x-axis values at which the bus `myBus` transitions from binary value `0b11` ($3_{decimal}$) to binary value `0b00` ($0_{decimal}$) because *n*=`0`.

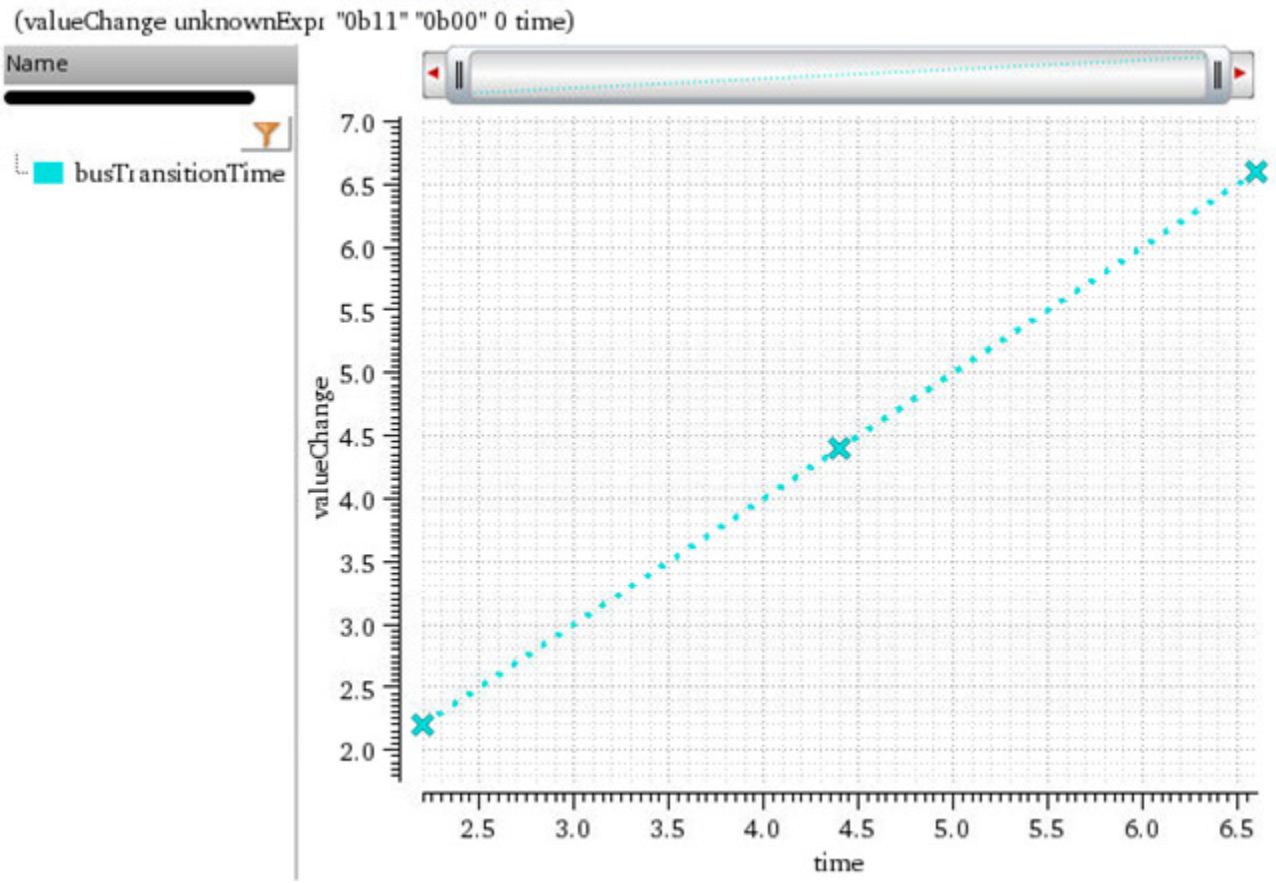
Note that the x-axis unit of the resulting waveform, which is stored in a variable `allTransitions`, is time.

```
allTransitions=busTransition(myBus "0b11" "0b00" 0 'time)
=> srrWave:0x31451030
```

The following example plots the waveform represented by `allTransitions`.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(allTransitions)
                ?expr list("busTransitionTime")
                ?color list("y12")
                ?lineStyle list("dot")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("x")
                )
```

```
=> t
```



The following example returns all x-axis values at which the bus `myBus` transitions from binary value `0b11` ($3_{decimal}$) to binary value `0b00` ($0_{decimal}$) because *n*=`0`.

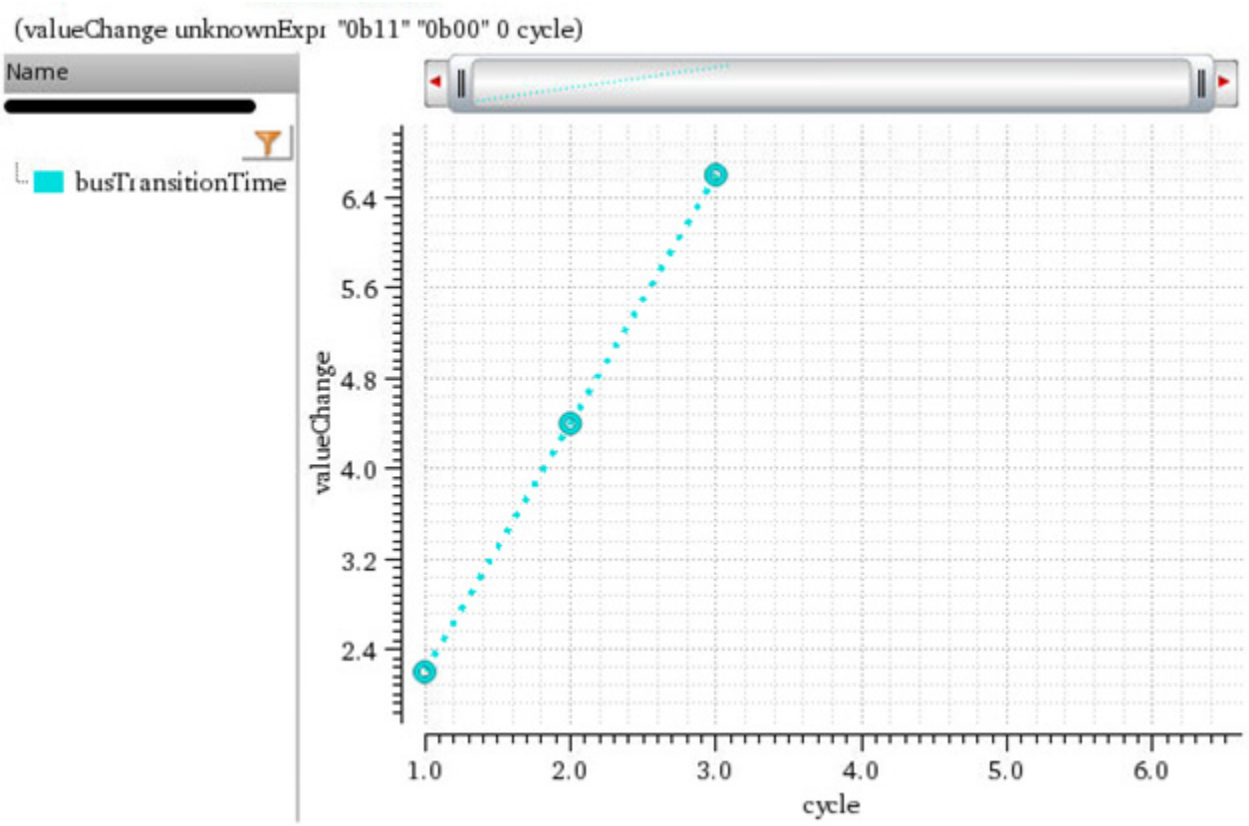
Note that the x-axis unit of the resulting waveform, which is stored in a variable `allTransitionsC`, is `cycle`.

```
allTransitionsC=busTransition(myBus "0b11" "0b00" 0 'cycle)
=> srrWave:0x31451040
```

The following example plots the waveform represented by `allTransitionsC`.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(allTransitionsC)
                ?expr list("busTransitionTime")
                ?color list("y12")
                ?lineStyle list("dot")
                ?lineThickness list("thick")
```

```
                ?showSymbols list(t)
                ?dataSymbol list("O")
                )
=> t
```



calCalcInput

```
calCalcInput(
     l_keywords
     [ t_expression ]
     )
     => t / nil
```

Description

Manipulates Buffer and Stack contents and enters arbitrary expressions into Buffer.

Arguments

l_keywords	Integer, symbol, or list of symbols indicating the keywords whose corresponding function is to be performed on the contents of Buffer or Stack.
t_expression	Expression to be copied to Buffer content.

Arguments

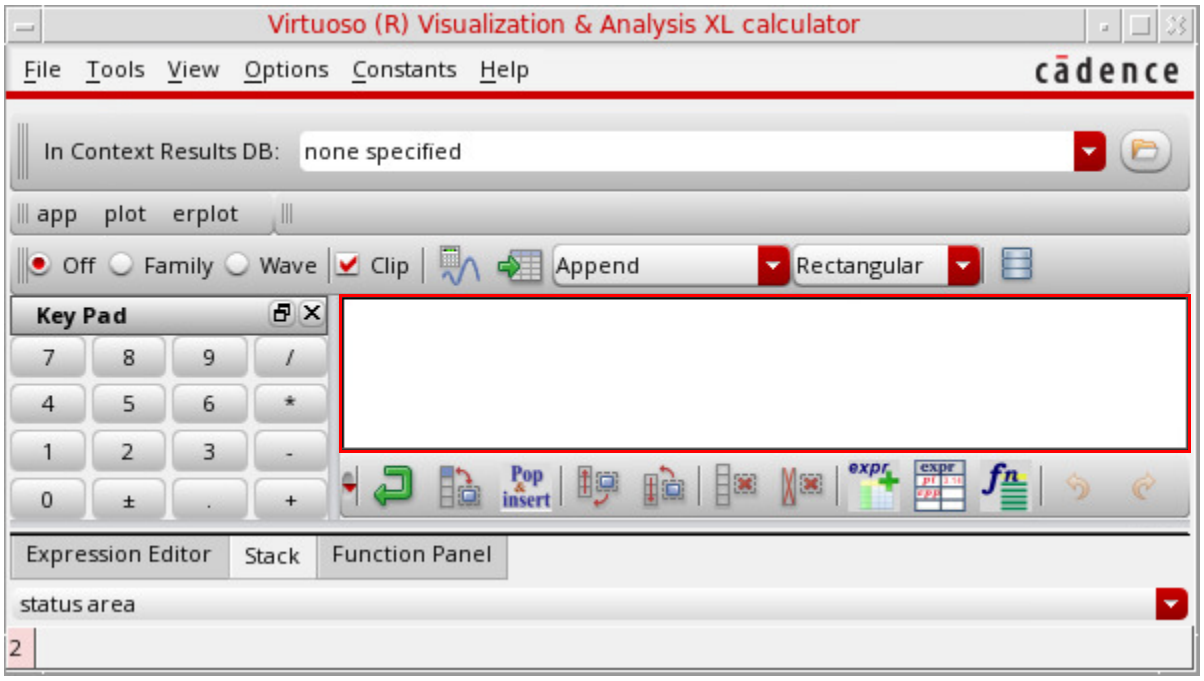
t	The command executes successfully.
nil	Indicates an error.

Examples

The following example opens Calculator.

```
calCalculatorFormCB()
=> t
```

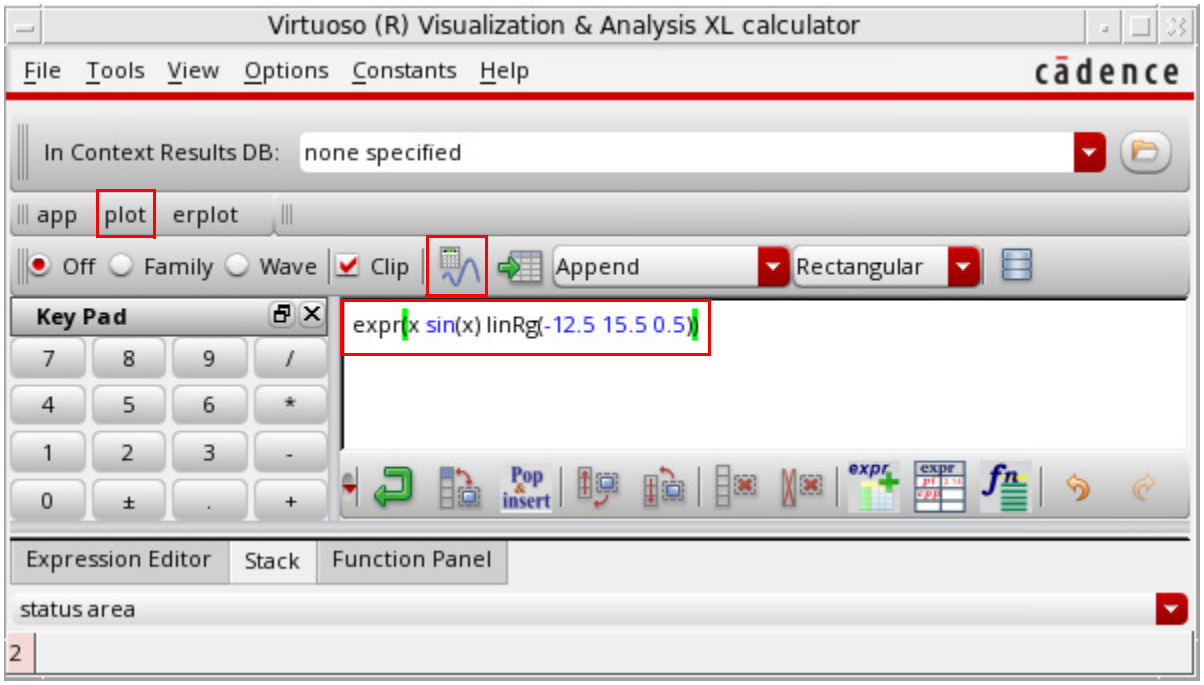
Note that the Buffer is currently empty.



The following example copies the expression "`expr(x sin(x) linRg(-12.5 15.5 0.5))`" in Buffer.

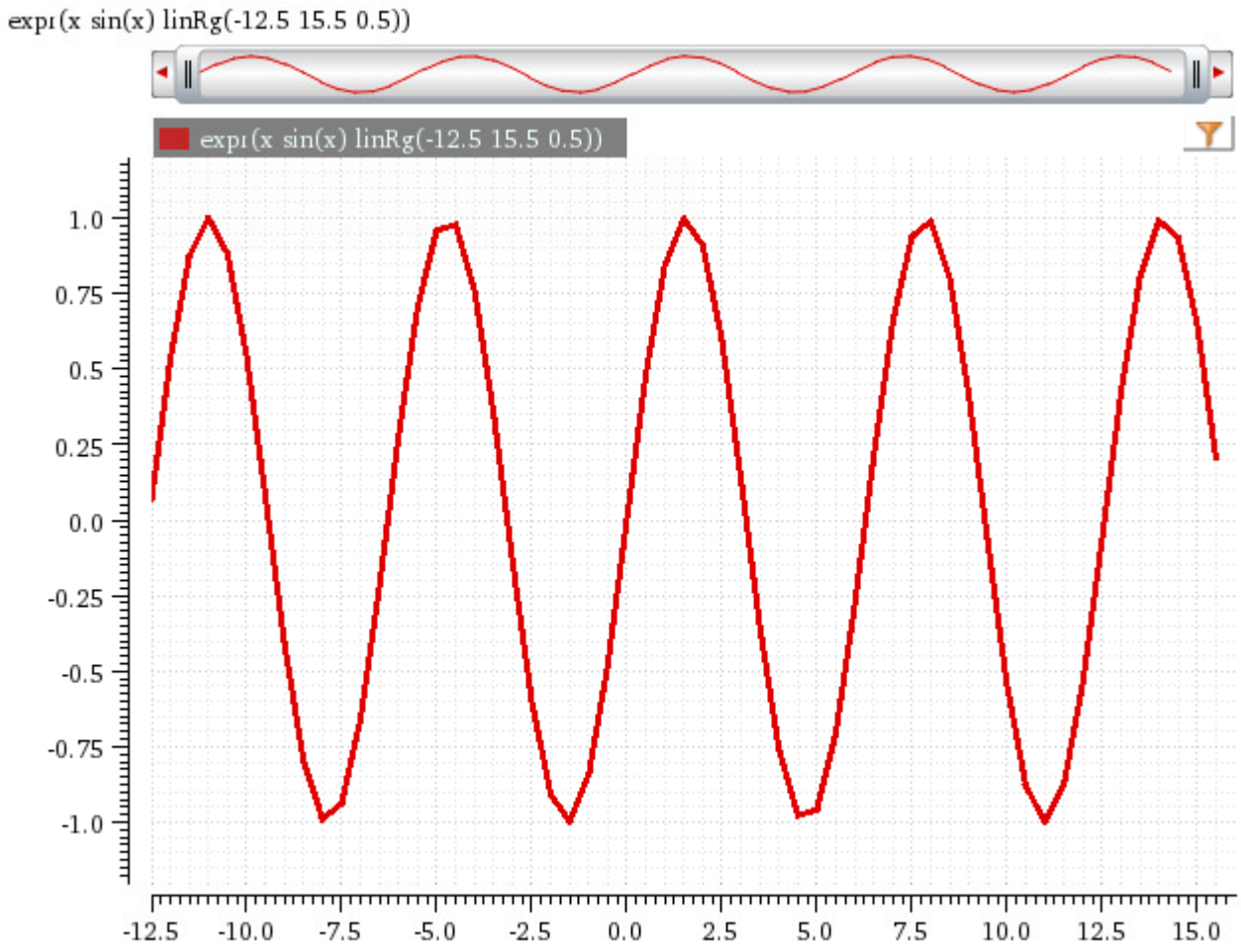
```
calCalcInput("expression" "expr(x sin(x) linRg(-12.5 15.5 0.5))")
=> calSetBuffer("expr(x sin(x) linRg(-12.5 15.5 0.5))")
=> t
```

```
=> t
```



Click *plot* or the *Evaluate the Buffer* icon to evaluate the expression in Buffer.

The resultant waveform is plotted in a Waveform window.



The following example clears the contents of Buffer.

```
calCalcInput("clear")
calSetBuffer("")
=> t
=> t
```

The following example places the expression "1+2" in Buffer.

```
calCalcInput("expression" "1+2")
calSetBuffer("1+2")
=> t
=> t
```

The following expression is created in Buffer.

1 + 2

The following example applies the function on the contents of Buffer in the order their corresponding keywords are specified.

```
calCalcInput(list("exp" "square" "ln"))
=> calSetBuffer("exp(1+2)")
=> t
=> calSetBuffer("exp(1+2)**2")
=> t
=> calSetBuffer("ln(exp(1+2)**2)")
=> t
=> t
```

The final expression created in Buffer is shown, as follows:

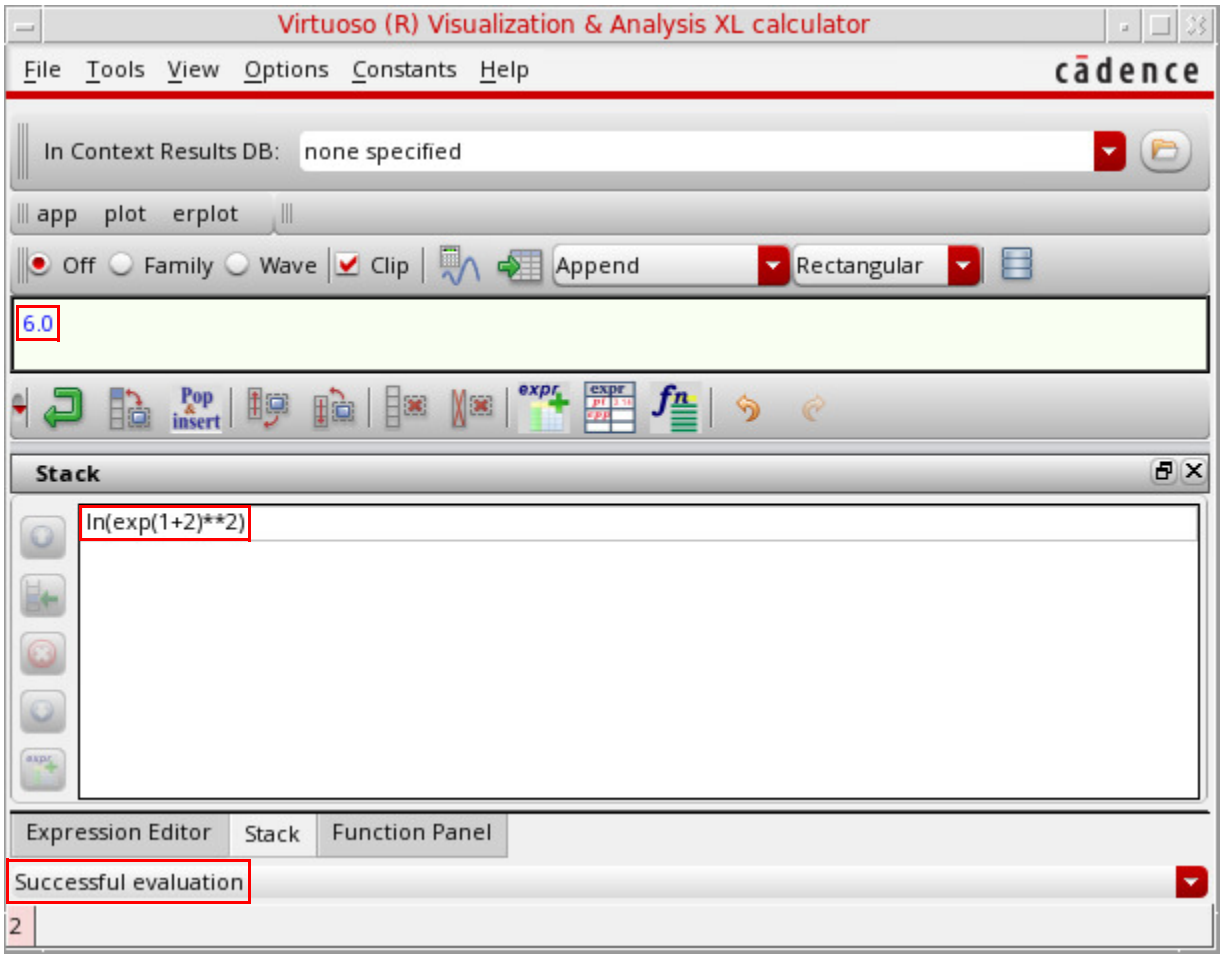
ln(**expr**(**1 + 2**)****2**)

If you evaluate this expression manually, the result is `6`.

$$\log_e\left((e^{(1+2)})^2\right) = \log_e\left((e^3)^2\right) = \log_e(e^6) = 6 \times \log_e e = 6 \times 1 = 6$$

Click *plot* or the *Evaluate the Buffer*  icon to evaluate the expression in the Buffer.

The resultant scalar value `6` is displayed in Buffer, as follows.



calCalculatorFormCB

```
calCalculatorFormCB(
     [ ?bBoxSpec t_bBoxSpec ]
     [ ?iconPosition t_iconPosition ]
     )
     => t / nil
```

Description

Opens Calculator, if not already open. If Calculator is open, the function brings it in the focus.

Arguments

`?bBoxSpec` *`t_bBoxSpec`*

Specification of box.

`?iconPosition` *`t_iconPosition`*

Position of icons in Calculator.

Value Returned

`t`	Calculator opens.
`nil`	Calculator does not open because of an error.

Examples

The following example opens Calculator and brings it in the focus.

```
calCalculatorFormCB()
=> t
```

calCreateSpecialFunction

```
calCreateSpecialFunction(
     [ ?formSym s_formSymbol ]
     [ ?formInitProc s_formInitializeProcedure ]
     [ ?formTitle t_formTitle ]
     [ ?formCallback t_formCallback ]
     [ ?envGetVal t_envGetVal ]
     )
     => nil
```

Description

Encapsulates the initialization and display of forms defined for a special (user-defined) function.

Arguments

?formSym *s_formSymbol*

Symbol of the form.

?formInitProc *s_formInitializeProcedure*

Symbol of 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.

Value Returned

t — The form for the special function is initialized and displayed.

`nil`	Indicates an error.

Examples

The following example defines a form `myForm` for a user-defined function. The form is represented by a symbol `specialForm`.

This form takes four arguments: *Start*, *Stop*, *Threshold*, and *Edge Type*.

These arguments are represented by symbols `a`, `b`, `c`, and `d`, respectively.

The first three arguments are string fields and the fourth argument *Edge Type* is a cyclic field. Valid values for the argument *Edge Type* are `Rising` and `Falling`. The default value is `Rising`.

```
procedure(myForm()
    let((a b c d fieldList)
        a=ahiCreateStringField(
            ?name 'a
            ?prompt "Start"
            ?value ""
        )
        b=ahiCreateStringField(
            ?name 'b
            ?prompt "Stop"
            ?value ""
        )
        c=ahiCreateStringField(
            ?name 'c
            ?prompt "Threshold"
            ?value ""
            )
        d=ahiCreateCyclicField(
            ?name 'd
            ?prompt "Edge Type"
            ?choices '("Rising" "Falling")
            ?value "Rising"
            ?defValue "Rising"
        )
        fieldList=list(
            list(a 20:10 230:10 90)
            list(b 20:40 230:10 90)
```

```
            list(c 20:70 230:10 90)
            list(d 20:110 230:10 90)
        )
    calCreateSpecialFunctionsForm('specialForm fieldList)
   )
)
=> myForm
```

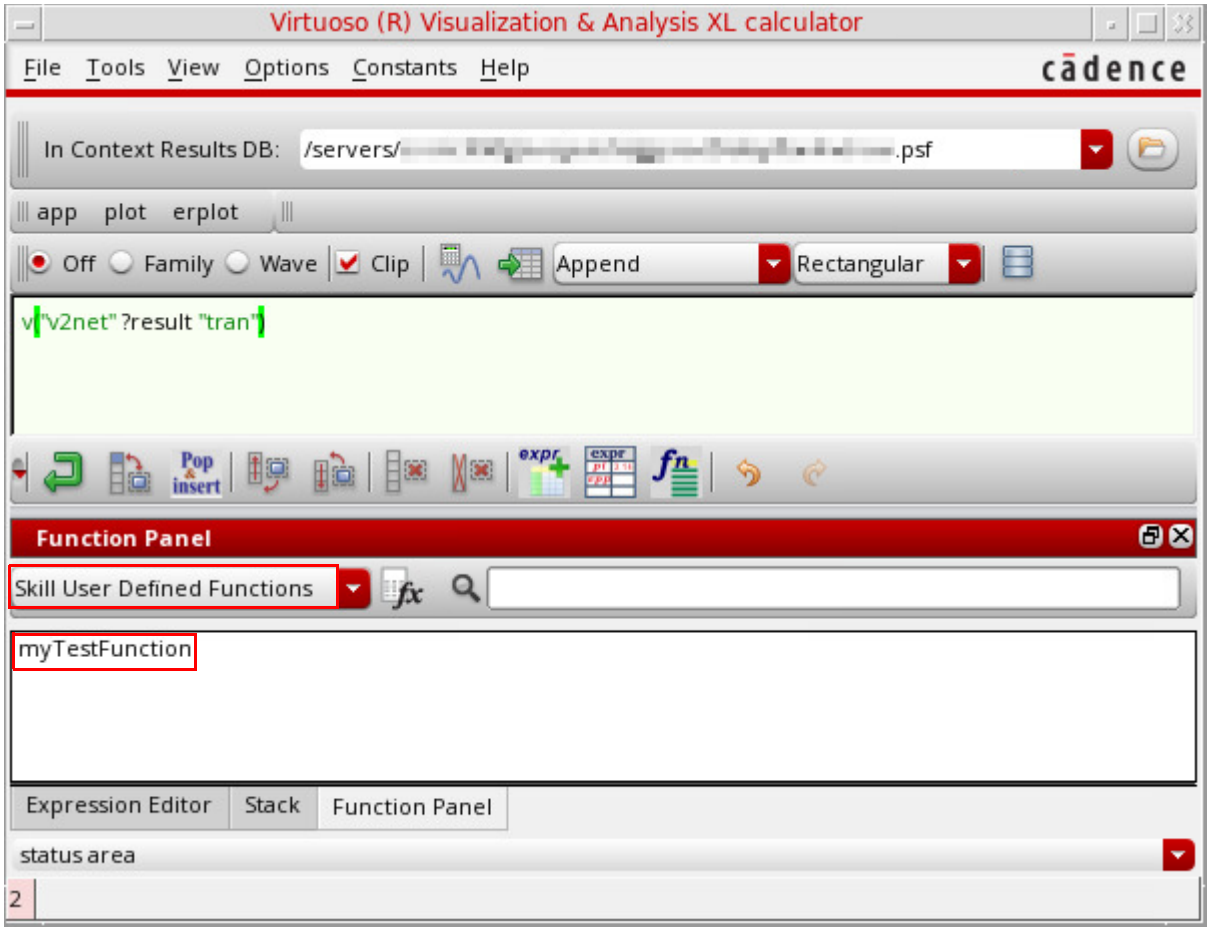
The following example defines a callback procedure `myFunctionPlotCB` that calls the function `calCreateSpecialFunction`, which creates and displays the form *My Test Function Form* created for the user-defined function. The user-defined function is represented by a symbol `myTestFunction`.

```
procedure(myFunctionPlotCB()
    calCreateSpecialFunction(
        ?formSym 'specialForm
        ?formInitProc 'myForm
        ?formTitle "My Test Function Form"
        ?formCallback "calSpecialFunctionInput('myTestFunction '( a b c d))"
    )
)
=> myFunctionPlotCB
```

The following example registers the function `myTestFunction` with its callback `myFunctionPlotCB`.

```
calRegisterSpecialFunction(
    list("myTestFunction" 'myFunctionPlotCB)
)
=> nil
```

The next time you open the Calculator, the function `myTestFunction`, which you defined using the function `calCreateSpecialFunction` appear in the *SKILL User Defined Functions* category of the *Function Panel*, as shown in the following figure.

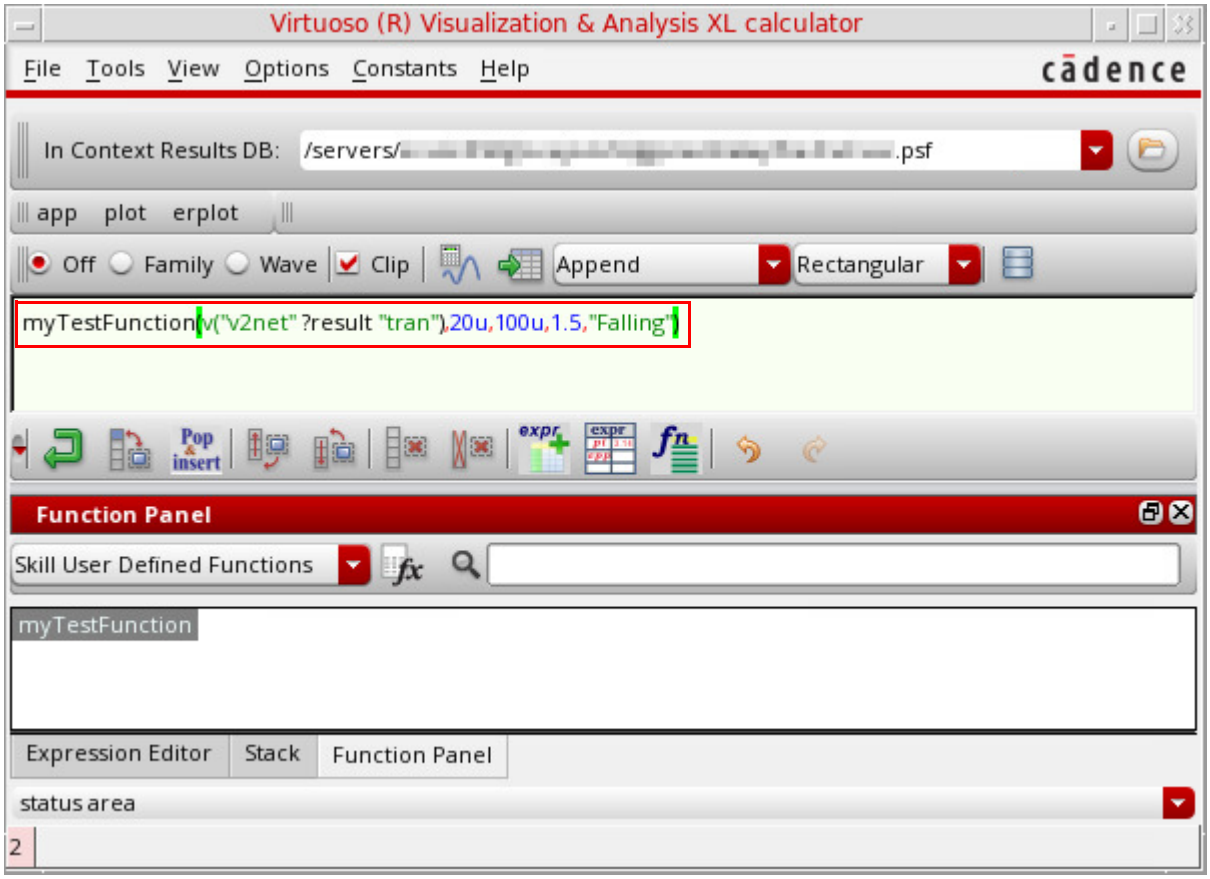


When you click *myTestFunction* from the *SKILL User Defined Functions* category in the *Function Panel*, the form that you created for the user-defined function using the function `calCreateSpecialFunctionsForm` opens.

Specify the arguments *Start*, *Stop*, *Threshold*, and *Edge Type* to `20u`, `100u`, `1.5`, and `Falling`, respectively and then click *OK*.

The following expression is built in the buffer with the specified form fields.

```
myTestFunction(v("v2net" ?result "tran"),20u,100u,1.5,"Falling")
```



calCreateSpecialFunctionsForm

```
calCreateSpecialFunctionsForm(
     s_formSymbol
     l_fieldList
     )
     => t / nil
```

Description

Registers and creates the form for a special (user-defined) function. The form title and callback are specified through the call to `calCreateSpecialFunction`.

The form is also created with `?unmapAfterCB` set to `t`.

Arguments

s_formSymbol	Symbol of the form.
l_fieldList	List of fields to be added in the form.

Value Returned

`t`	The form for the special function is created successfully.
`nil`	The form for the special function cannot be created because of an error.

Examples

The following example defines a form `myForm` for a user-defined function. The form is represented by a symbol `specialForm`.

This form takes four arguments: *Start*, *Stop*, *Threshold*, and *Edge Type*.

These arguments are represented by symbols `a`, `b`, `c`, and `d`, respectively.

The first three arguments are string fields and the fourth argument *Edge Type* is a cyclic field. Valid values for the argument *Edge Type* are `Rising` and `Falling`. The default value is `Rising`.

```
procedure(myForm()
    let((a b c d fieldList)
```

```
        a=ahiCreateStringField(
            ?name 'a
            ?prompt "Start"
            ?value ""
        )
        b=ahiCreateStringField(
            ?name 'b
            ?prompt "Stop"
            ?value ""
        )
        c=ahiCreateStringField(
            ?name 'c
            ?prompt "Threshold"
            ?value ""
            )
        d=ahiCreateCyclicField(
            ?name 'd
            ?prompt "Edge Type"
            ?choices '("Rising" "Falling")
            ?value "Rising"
            ?defValue "Rising"
        )
        fieldList=list(
            list(a 20:10 230:10 90)
            list(b 20:40 230:10 90)
            list(c 20:70 230:10 90)
            list(d 20:110 230:10 90)
        )
    calCreateSpecialFunctionsForm('specialForm fieldList)
   )
)
=> myForm
```

The following example defines a callback procedure `myFunctionPlotCB` that calls the function `calCreateSpecialFunction`, which creates and displays the form *My Test Function Form* created for the user-defined function. The user-defined function is represented by a symbol `myTestFunction`.

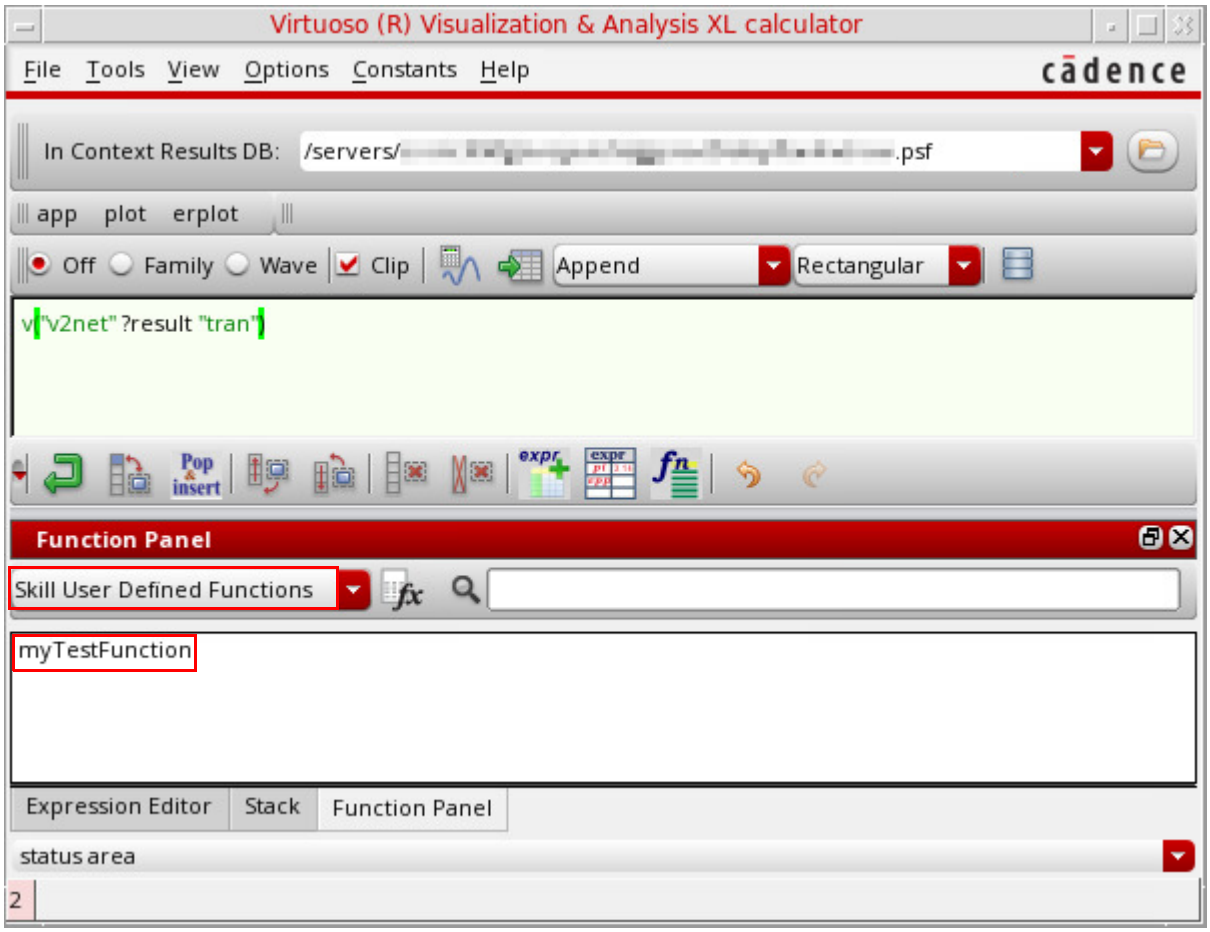
```
procedure(myFunctionPlotCB()
    calCreateSpecialFunction(
        ?formSym 'specialForm
        ?formInitProc 'myForm
```

```
        ?formTitle "My Test Function Form"
        ?formCallback "calSpecialFunctionInput('myTestFunction '( a b c d))"
    )
)
=> myFunctionPlotCB
```

The following example registers the function `myTestFunction` with its callback `myFunctionPlotCB`.

```
calRegisterSpecialFunction(
    list("myTestFunction" 'myFunctionPlotCB)
)
=> nil
```

The next time you open the Calculator, the function `myTestFunction`, which you defined using the function `calCreateSpecialFunction` appear in the *SKILL User Defined Functions* category of the *Function Panel*, as shown in the following figure.

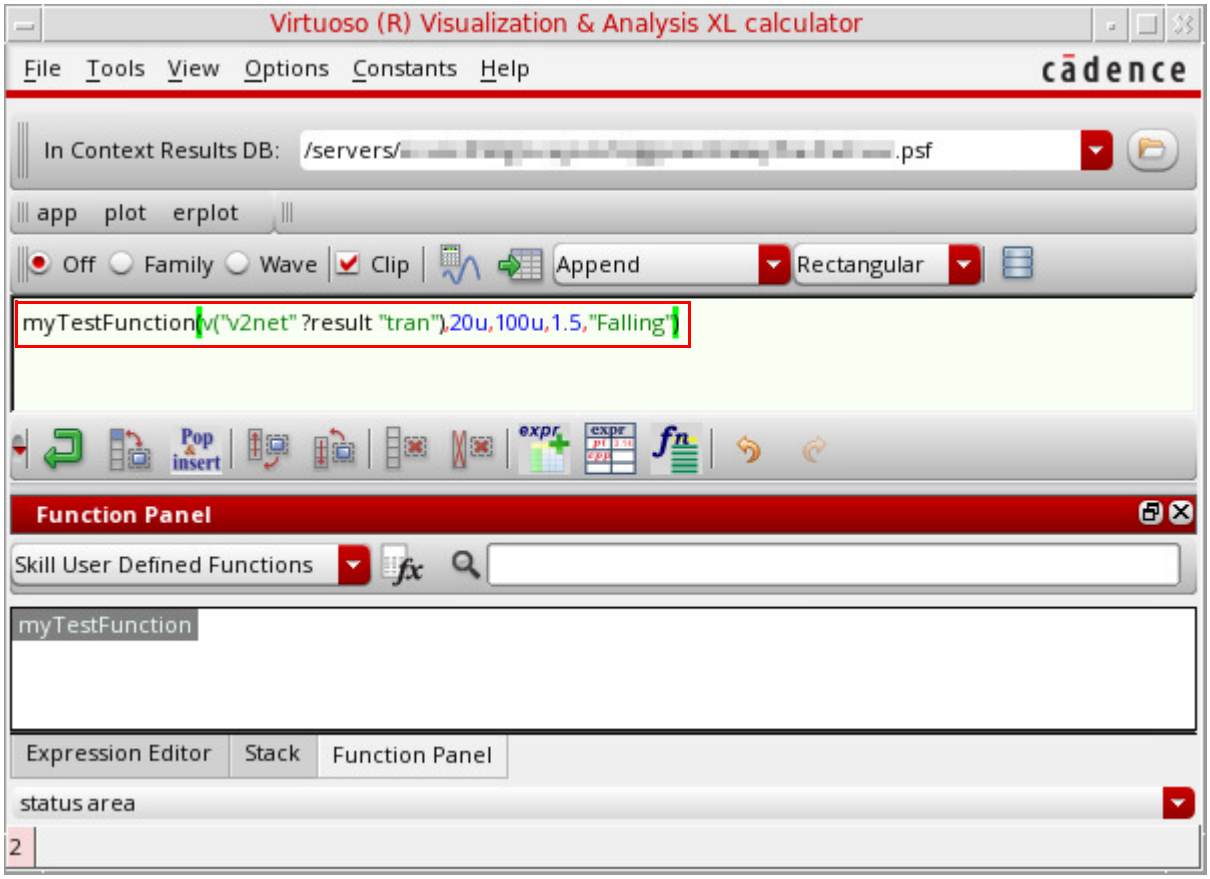


When you click *myTestFunction* from the *SKILL User Defined Functions* category in the *Function Panel*, the form that you created for the user-defined function using the function `calCreateSpecialFunctionsForm` opens.

Specify the arguments *Start*, *Stop*, *Threshold*, and *Edge Type* to `20u`, `100u`, `1.5`, and `Falling`, respectively and then click *OK*.

The following expression is built in the buffer with the specified form fields.

```
myTestFunction(v("v2net" ?result "tran"),20u,100u,1.5,"Falling")
```



calGetBuffer

```
calGetBuffer(
     )
     => t_bufferExpression / nil
```

Description

Returns the expression that is currently stored in calculator Buffer.

Arguments

None

Value Returned

t_bufferExpression	Expression set in Buffer.
nil	No expression is currently set in the Buffer or there is an error.

Examples

The following example sets the specified expression in the Calculator Buffer.

```
calSetBuffer("expr(x sin(x) linRg(2 4 0.5))")
=> t
```

The following example returns the expression that is currently set in the Calculator Buffer.

```
calGetBuffer()
=> "expr(x sin(x) linRg(2 4 0.5))"
```

calRegisterSpecialFunction

```
calRegisterSpecialFunction(
     l_specialFunctionInfo
     )
     => l_specialFunctionInfo / nil
```

Description

Registers the specified information about the special (user-defined) function, if it is not already registered.

Arguments

l_specialFunctionInfo

List specifying the following information about the user-defined function:

- *t_specialFunctionName*: Name of the user-defined function that appears in the *Skill User Defined Functions* list in the *Function Panel* of Calculator.
- *s_specialFunctionCallback*: Symbol of the callback procedure to the user-defined function.

Value Returned

l_specialFunctionInfo

Name and callback procedure set for the user-defined function.

`nil` The user-defined function cannot be registered because of an error.

Examples

The following example defines a form `myForm` for a user-defined function. The form is represented by a symbol `specialForm`.

This form takes four arguments: *Start*, *Stop*, *Threshold*, and *Edge Type*.

These arguments are represented by symbols `a`, `b`, `c`, and `d`, respectively.

The first three arguments are string fields and the fourth argument *Edge Type* is a cyclic field. Valid values for the argument *Edge Type* are `Rising` and `Falling`. The default value is `Rising`.

```
procedure(myForm()
    let((a b c d fieldList)
        a=ahiCreateStringField(
            ?name 'a
            ?prompt "Start"
            ?value ""
        )
        b=ahiCreateStringField(
            ?name 'b
            ?prompt "Stop"
            ?value ""
        )
        c=ahiCreateStringField(
            ?name 'c
            ?prompt "Threshold"
            ?value ""
            )
        d=ahiCreateCyclicField(
            ?name 'd
            ?prompt "Edge Type"
            ?choices '("Rising" "Falling")
            ?value "Rising"
            ?defValue "Rising"
        )
        fieldList=list(
            list(a 20:10 230:10 90)
            list(b 20:40 230:10 90)
            list(c 20:70 230:10 90)
            list(d 20:110 230:10 90)
        )
    calCreateSpecialFunctionsForm('specialForm fieldList)
   )
)
=> myForm
```

The following example defines a callback procedure `myFunctionPlotCB` that calls the function `calCreateSpecialFunction`, which creates and displays the form *My Test*

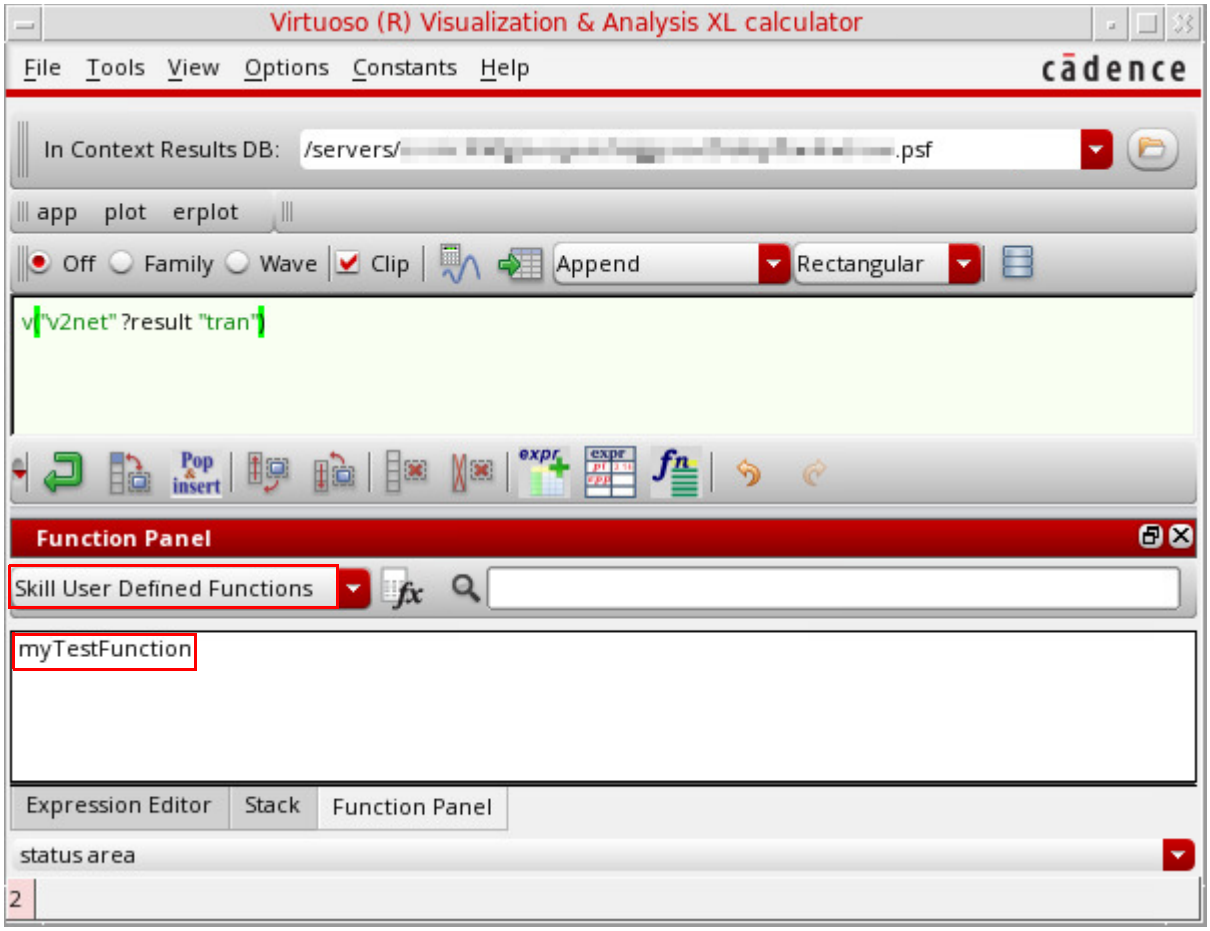
Function Form created for the user-defined function. The user-defined function is represented by a symbol `myTestFunction`.

```
procedure(myFunctionPlotCB()
    calCreateSpecialFunction(
        ?formSym 'specialForm
        ?formInitProc 'myForm
        ?formTitle "My Test Function Form"
        ?formCallback "calSpecialFunctionInput('myTestFunction '( a b c d))"
    )
)
=> myFunctionPlotCB
```

The following example registers the function `myTestFunction` with its callback `myFunctionPlotCB`.

```
calRegisterSpecialFunction(
    list("myTestFunction" 'myFunctionPlotCB)
)
=> nil
```

The next time you open the Calculator, the function `myTestFunction`, which you defined using the function `calCreateSpecialFunction` appear in the *SKILL User Defined Functions* category of the *Function Panel*, as shown in the following figure.

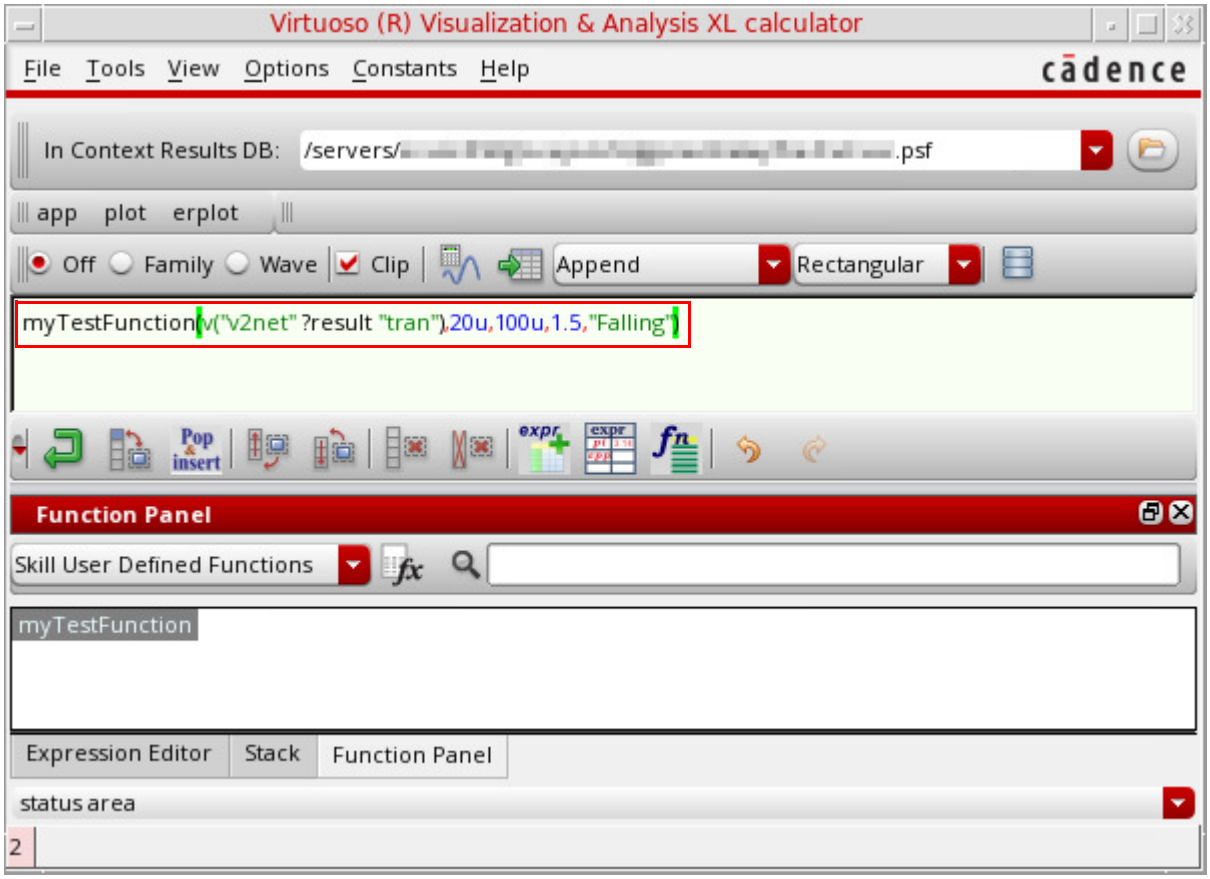


When you click *myTestFunction* from the *SKILL User Defined Functions* category in the *Function Panel*, the form that you created for the user-defined function using the function `calCreateSpecialFunctionsForm` opens.

Specify the arguments *Start*, *Stop*, *Threshold*, and *Edge Type* to `20u`, `100u`, `1.5`, and `Falling`, respectively and then click *OK*.

The following expression is built in the buffer with the specified form fields.

```
myTestFunction(v("v2net" ?result "tran"),20u,100u,1.5,"Falling")
```



calSetCurrentTest

```
calSetCurrentTest(
     t_testName
     )
     => nil
```

Description

Specifies the name of the test that you want to set as the current test in the Calculator. When an access function, such as `vt`, is used from the Calculator, it displays the schematic associated with the current test.

If the *Test Selection Toolbar* command is selected, the available tests are displayed in a drop-down list in the Calculator. You can also use this drop-down list to change the current test.

Arguments

t_testName	Name of the test to be set as the current test.

Value Returned

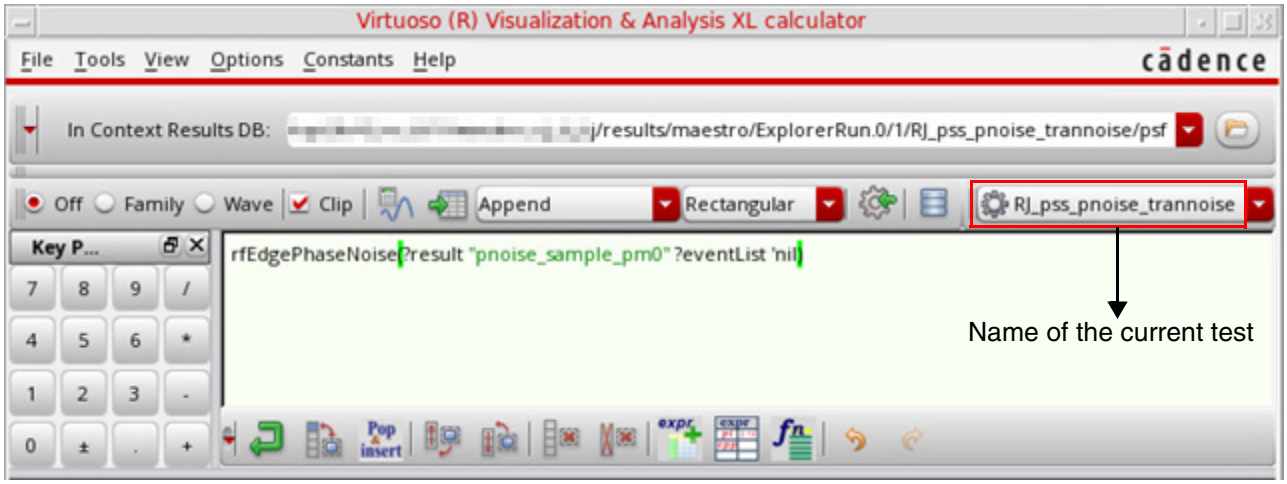
`t`	The specified test is set as the current test in the Calculator.
`nil`	The specified test cannot be set as the current test because of an error.

Examples

The following example sets the current test to `RJ_pss_pnoise_trannoise` in the Calculator.

```
calSetCurrentTest("RJ_pss_pnoise_trannoise")
```

```
=> nil
```



calSpecialFunctionInput

```
calSpecialFunctionInput(
     s_specialFunctionName
     l_formFields
     )
     => t_expression / nil
```

Description

Checks the buffer and stack and processes the arguments defined under *l_formFields* into the buffer expression.

Arguments

s_specialFunctionName

Symbol of the user-defined function.

l_formFields

Ordered list of form field symbols that composes the argument list of the user-defined function.

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	Expression is set in the Buffer.
`nil`	Indicates an error.

Examples

The following example defines a form `myForm` for a user-defined function. The form is represented by a symbol `specialForm`.

This form takes four arguments: *Start*, *Stop*, *Threshold*, and *Edge Type*.

These arguments are represented by symbols `a`, `b`, `c`, and `d`, respectively.

The first three arguments are string fields and the fourth argument *Edge Type* is a cyclic field. Valid values for the argument *Edge Type* are `Rising` and `Falling`. The default value is `Rising`.

```
procedure(myForm()
    let((a b c d fieldList)
        a=ahiCreateStringField(
            ?name 'a
            ?prompt "Start"
            ?value ""
        )
        b=ahiCreateStringField(
            ?name 'b
            ?prompt "Stop"
            ?value ""
        )
        c=ahiCreateStringField(
            ?name 'c
            ?prompt "Threshold"
            ?value ""
            )
        d=ahiCreateCyclicField(
            ?name 'd
            ?prompt "Edge Type"
            ?choices '("Rising" "Falling")
            ?value "Rising"
            ?defValue "Rising"
        )
        fieldList=list(
            list(a 20:10 230:10 90)
            list(b 20:40 230:10 90)
            list(c 20:70 230:10 90)
            list(d 20:110 230:10 90)
        )
    calCreateSpecialFunctionsForm('specialForm fieldList)
   )
)
=> myForm
```

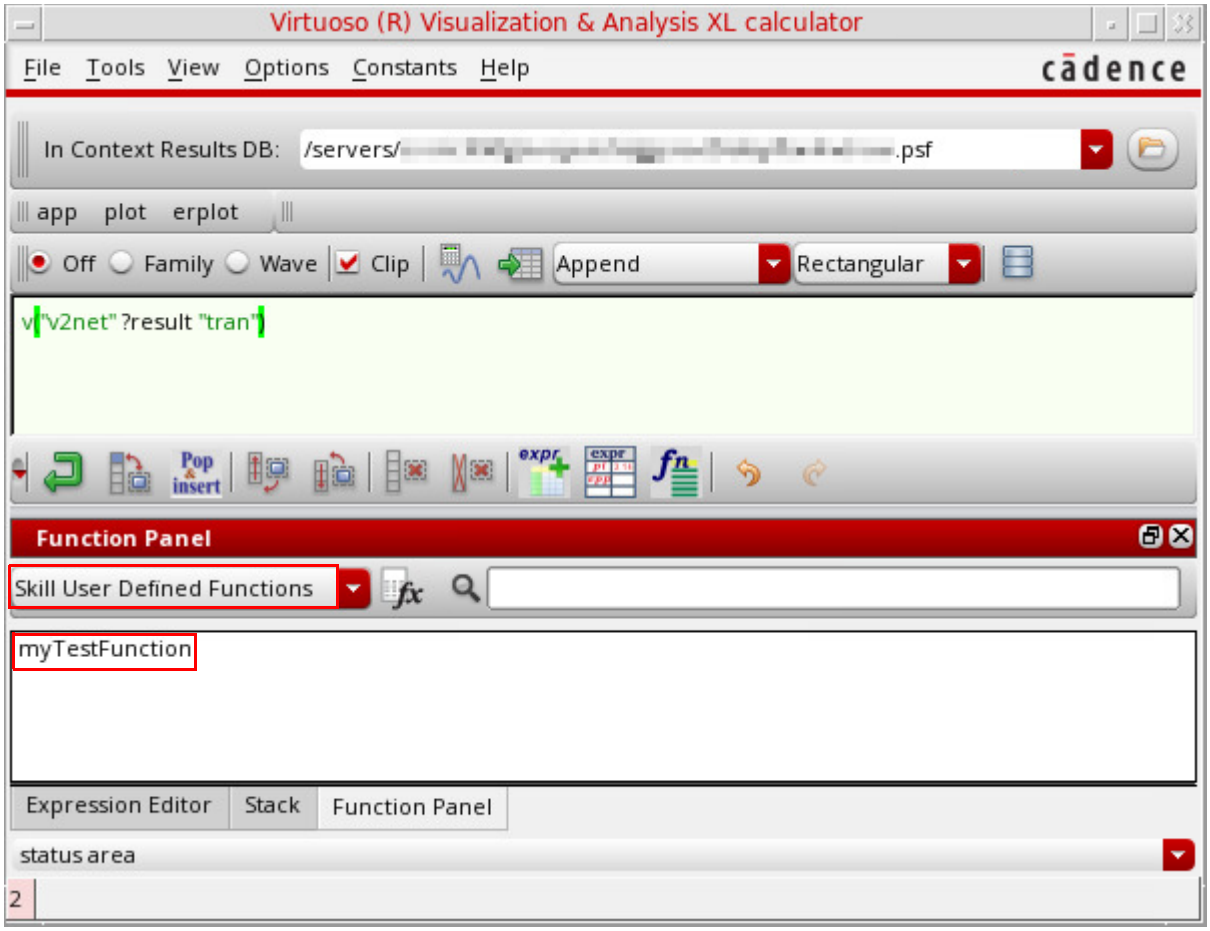
The following example defines a callback procedure `myFunctionPlotCB` that calls the function `calCreateSpecialFunction`, which creates and displays the form *My Test Function Form* created for the user-defined function. The user-defined function is represented by a symbol `myTestFunction`.

```
procedure(myFunctionPlotCB()
    calCreateSpecialFunction(
        ?formSym 'specialForm
        ?formInitProc 'myForm
        ?formTitle "My Test Function Form"
        ?formCallback "calSpecialFunctionInput('myTestFunction '( a b c d))"
    )
)
=> myFunctionPlotCB
```

The following example registers the function `myTestFunction` with its callback `myFunctionPlotCB`.

```
calRegisterSpecialFunction(
    list("myTestFunction" 'myFunctionPlotCB)
)
=> nil
```

The next time you open the Calculator, the function `myTestFunction`, which you defined using the function `calCreateSpecialFunction` appear in the *SKILL User Defined Functions* category of the *Function Panel*, as shown in the following figure.

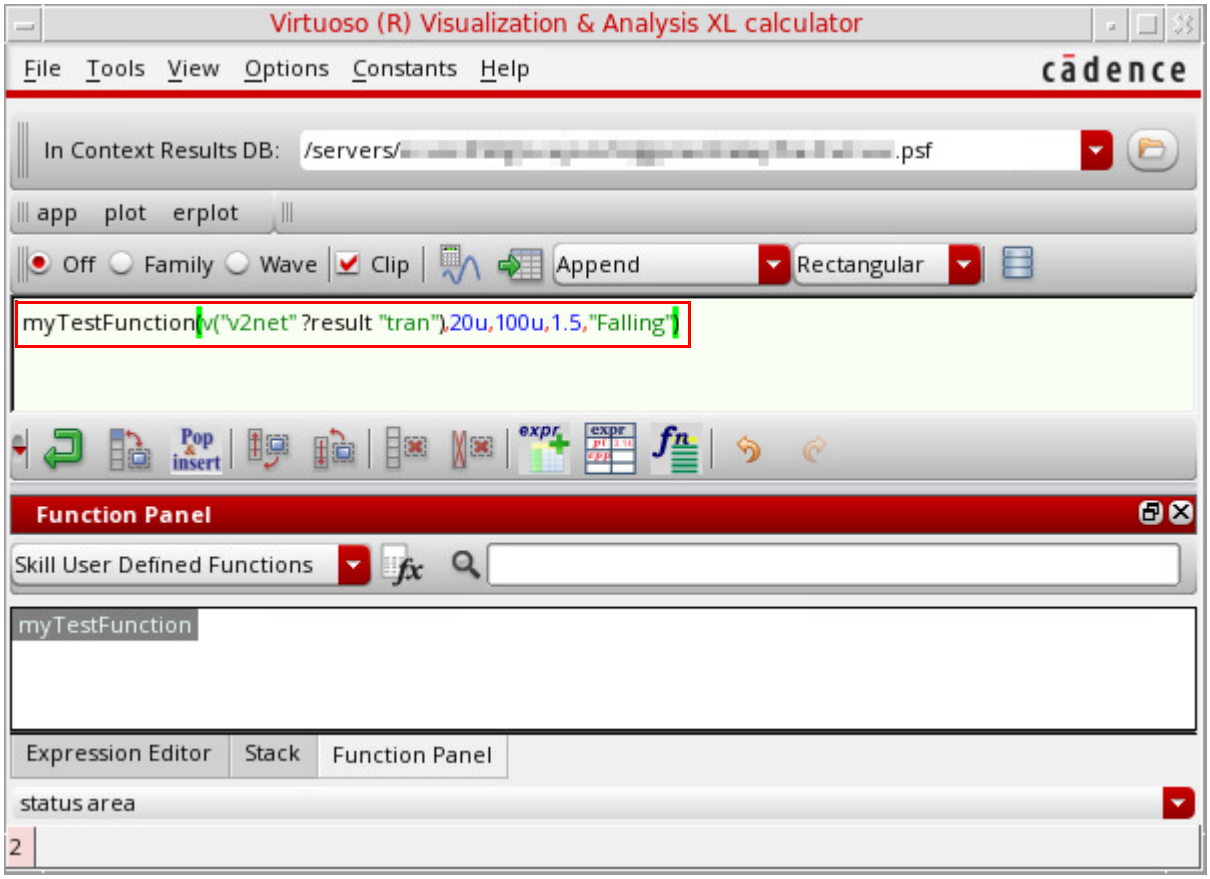


When you click *myTestFunction* from the *SKILL User Defined Functions* category in the *Function Panel*, the form that you created for the user-defined function using the function `calCreateSpecialFunctionsForm` opens.

Specify the arguments *Start*, *Stop*, *Threshold*, and *Edge Type* to `20u`, `100u`, `1.5`, and `Falling`, respectively and then click *OK*.

The following expression is built in the buffer with the specified form fields.

```
myTestFunction(v("v2net" ?result "tran"),20u,100u,1.5,"Falling")
```



caliModeToggle

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

Description

Toggles Algebraic or RPN-specific buttons on the Calculator based on the currently set Calculator mode. Calculator works in two modes—*RPN Mode* and *Algebraic Mode*.

These modes provide different sets of operations for building and evaluating expressions. Default mode is *RPN Mode*.

Arguments

None

Value Returned

`t`	The command runs successfully.
`nil`	Indicates an error.

Examples

The following example toggles Calculator mode.

```
caliModeToggle()
=> t
```

caliRestoreDefaultWindowSize

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

Description

Restores the original size of Calculator while maintaining the same top-left coordinate of the current window position.

Arguments

None

Value Returned

`t`	Original size of Calculator is restored.
`nil`	Original size of Calculator cannot be restored because of an error.

Examples

The following example restores the original size of Calculator.

```
caliRestoreDefaultWindowSize()
=> t
```

calSetBuffer

```
calSetBuffer(
     t_bufferExpression
     )
     => t / nil
```

Description

Sets the contents of the Buffer in Calculator.

Arguments

t_bufferExpression

Expression to be set in Buffer.

Value Returned

`t`	The expression is set in the Buffer.
`nil`	The expression cannot be set because of an error.

Examples

The following example sets the specified expression in the Calculator Buffer.

```
calSetBuffer("expr(x sin(x) linRg(2 4 0.5))")
=> t
```

The following example returns the expression that is currently set in the Calculator Buffer.

```
calGetBuffer()
=> "expr(x sin(x) linRg(2 4 0.5))"
```

dBm50ohm

```
dBm50ohm(
     { o_waveform | n_voltage }
     )
     => o_waveform | n_dBm
```

Description

Calculates output power in decibel-milliwatts (dBm) from the specified voltage (signal or value) for a 50 ohm resistance.

dBm is a unit of level that expresses power level in dB with reference to `1`mW. dBm is a dimensionless unit, but because it compares to a fixed reference value, the dBm is an absolute rating.

The power in dBm is calculated using the following formula:

$$\text{Power in dBm} = 10 \times \log_{10}\left(\frac{V^2}{R \times 1mW}\right)$$

$$\text{Power in dBm} = \frac{10 \times \log_e\left(\frac{V^2}{R \times 1mW}\right)}{\log_e 10}$$

Where V= voltage in volt, R= `50` ohm and, `1`mW= `0.001` watt

Arguments

`o_waveform` or *`n_voltage`*

Waveform of a voltage signal or the voltage value in volt.

Value Returned

`o_waveform`	Waveform representing the power output in dBm if the input is a waveform representing a voltage signal.
`n_dBm`	Scalar value indicating the power output in dBm if the input is a scalar voltage value.

Examples

The following example opens simulation results stored in the specified results directory.

```
openResults("./simulation/lib/cell/maestro/results/maestro/ExplorerRun.0/1/test/
psf")
=> "./simulation/lib/cell/maestro/results/maestro/ExplorerRun.0/1/test/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
=>
tran(tranOp model instance output designParamVals
    primitives subckts variables
)
```

The following example selects the result `tran`.

```
selectResults('tran)
=> stdobj@0x2ebd80c8
```

The following example lists the outputs available in the selected result.

```
outputs()
=> ("/net1" "/net2")
```

The following example creates a waveform object `v`, which represents the voltage waveform of `net1`.

```
v=v("net1")
=> srrWave:0x34a8f020
```

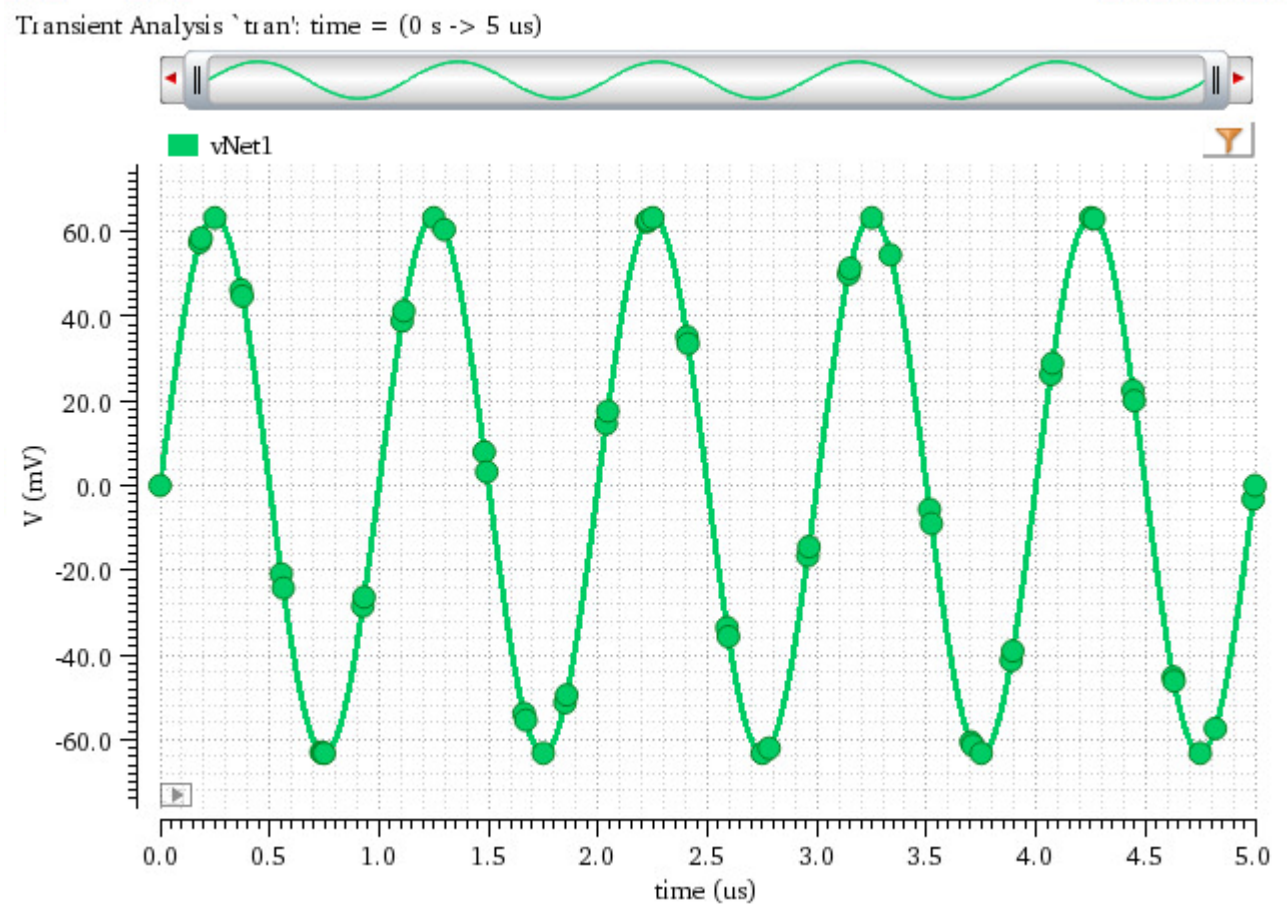
The following example creates a Waveform window `win1` and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example plots the waveform `v` in the Waveform window `win1`.

```
awvPlotWaveform(
                win1
                list(v)
                ?expr list("vNet1")
                ?color list("y18")
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list(".")
                )
```

```
=> t
```



The following example creates another Waveform window and returns it window ID.

```
win2=awvCreatePlotWindow()
=> window:4
```

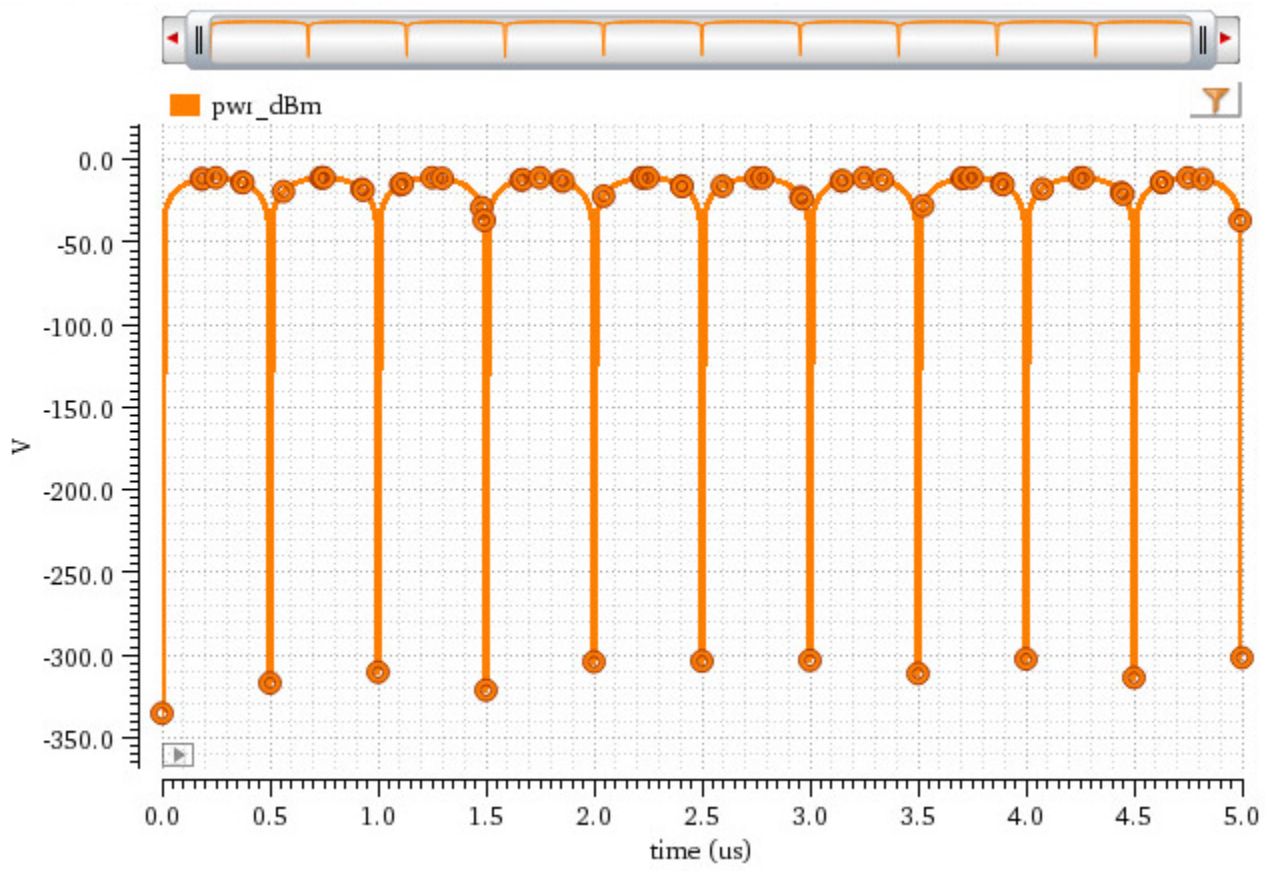
The following example creates a waveform object `p`, which represents the power in `dBm` for the specified voltage signal `v`.

```
p=dBm50ohm(v)
=> srrWave:0x34a8f030
```

The following example plots the waveform `p` in the Waveform window `win2`.

```
awvPlotWaveform(
                win2
                list(p)
                ?expr list("pwr_dBm")
                ?color list("y6")
                ?lineType list("line")
```

```
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("o")
                )
=> t
```



The following example calculates the power in `dBm` for the voltage value `10` volt for a `50` ohm impedance.

```
dBm50ohm(10)
=> 33.0103
```

dBm50ohmAny

```
dBm50ohmAny(
     n_voltage
     n_constant
     )
     => n_dBm
```

Description

Calculates output power in decibel-milliwatts (dBm) from the specified voltage value for a 50 ohm resistance.

dBm is a unit of level that expresses power level in dB with reference to 1mW. dBm is a dimensionless unit, but because it compares to a fixed reference value, the dBm is an absolute rating.

The power in dBm is calculated using the following formula:

$$\text{Power in dBm} = 10 \times \log_{10}\left(\frac{V^2}{R \times 1mW}\right)$$

$$\text{Power in dBm} = \frac{10 \times \log_{e}\left(\frac{V^2}{R \times 1mW}\right)}{\log_{e} 10}$$

Where, V= Voltage in volt, R= `50` ohm and, 1mW= `0.001` watt

Arguments

n_voltage	Voltage value in volt.
n_constant	Constant value.

Value Returned

n_dBm	Power output in dBm.

Examples

The following example calculates the power in dBm for the voltage value `10` volt and resistance value `50` ohm.

```
dBm50ohmAny(10 4.342945)
=> 33.0103
```

expr

```
expr(
     var
     expr
     l_values
     )
     => o_waveform / nil
```

Description

Evaluates the expression *expr* by setting each of the values in the *l_values* list to the *var* variable in the expression.

A waveform object is created with *l_values* forming x vectors and the evaluated *expr* forming y vectors.

Arguments

var	Variable.
expr	Expression containing the variable *var*.
l_values	List of values of x vectors.

Value Returned

o_waveform	Waveform object created by evaluating the expression *expr*.
nil	because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

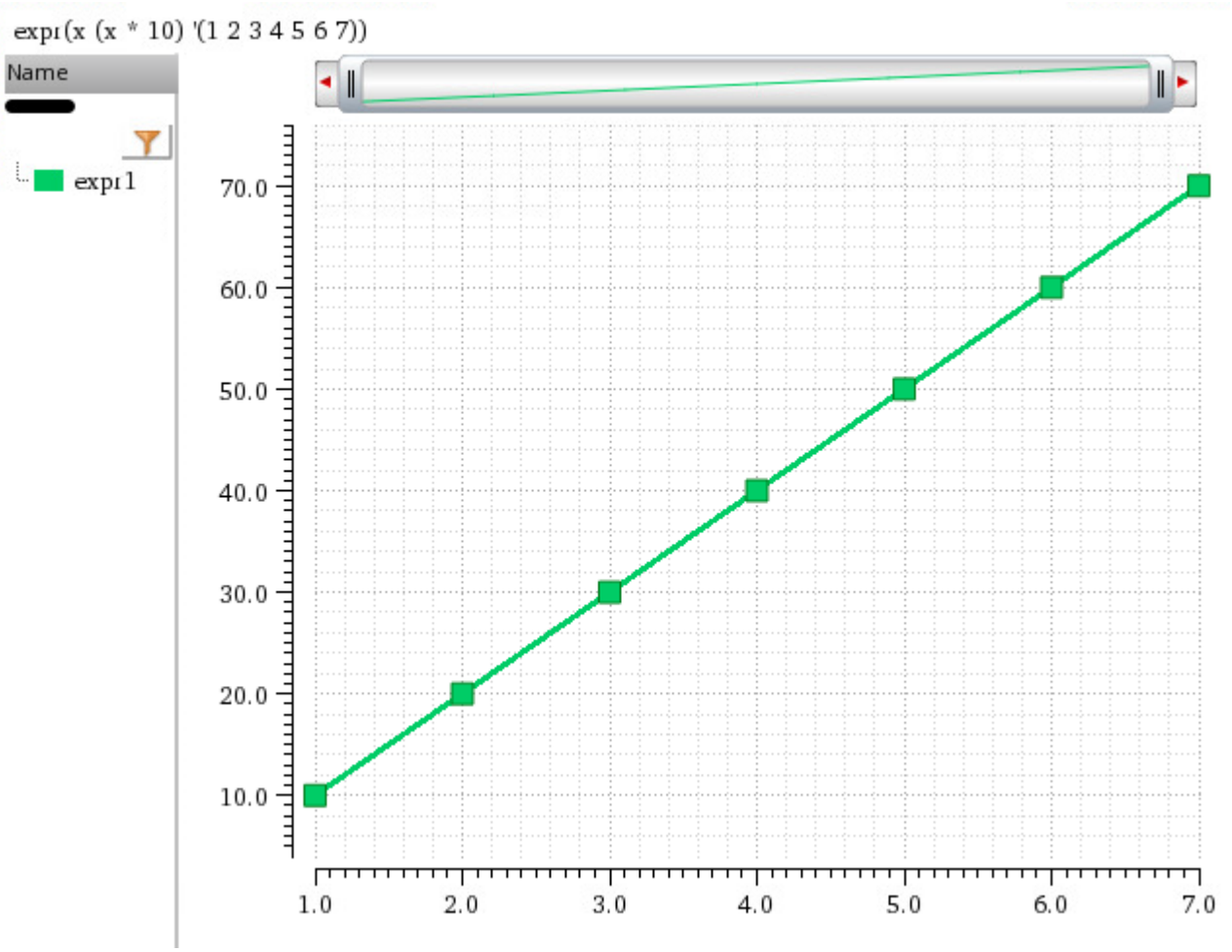
```
win=awvCreatePlotWindow()
=> window:3
```

The following example creates a waveform object wave1 by evaluating the expression *x**10 for different values of variable *x*. The expression is evaluated for different values of x vectors from 1 through 7.

```
wave1=expr(x (x*10) '(1 2 3 4 5 6 7))
=> srrWave:0x31180020
```

The following example plots the waveform object `wave1` in the specified Waveform window, which is created using the `awvCreatePlotWindow` function.

```
awvPlotWaveform(
                win
                list(wave1)
                ?expr list("expr1")
                ?color list("y18")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list(3)
                )
=> t
```



The following example creates another Waveform window and returns its Window ID.

```
win=awvCreatePlotWindow()
=> window:4
```

The following example creates a waveform object `wave2` by evaluating the expression `sin(x)` from *x*=`-12.5` to *x*=`15.5`.

The expression is evaluated at *x*=`-12.5`, `-12.0`, `-11.5`,...,`14.5`, `15.0`, and `15.5`.

```
wave2=expr(x sin(x) linRg(-12.5 15.5 0.5))
=> srrWave:0x30bfb020
```

The following example creates a waveform object `wave3` by evaluating the expression `-sin(x)` from *x*=`-12.5` to *x*=`15.5`.

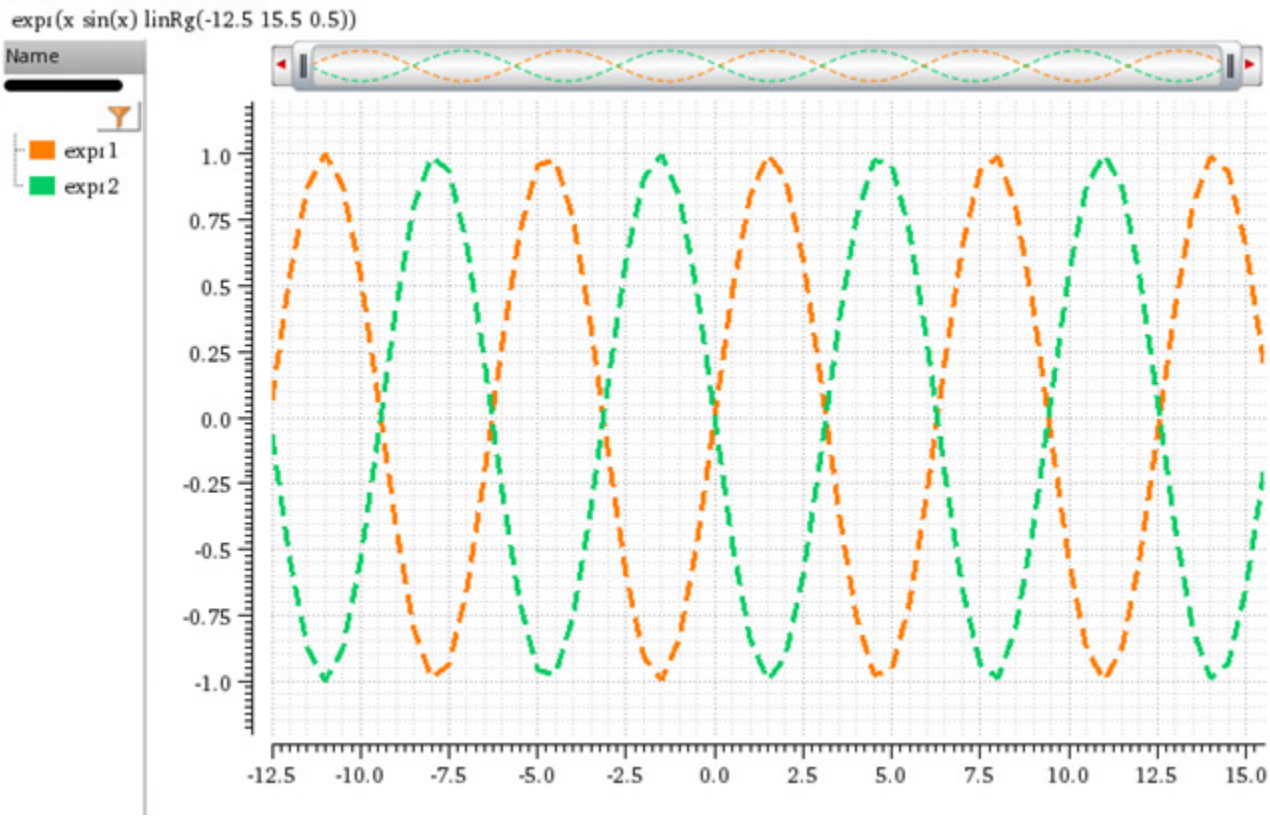
The expression is evaluated at *x*=`-12.5`, `-12.0`, `-11.5`,...,`14.5`, `15.0`, and `15.5`.

```
wave3=expr(x -sin(x) linRg(-12.5 15.5 0.5))
=> srrWave:0x30bfb030
```

The following example plots the waveform objects `wave2` and `wave3` in the specified Waveform window, which is created using the `awvCreatePlotWindow` function.

```
awvPlotWaveform(
                win
                list(wave2 wave3)
                ?expr list("expr1" "expr2")
                ?color list("y6" "y18")
                ?index list(1 2)
                ?lineType list("line" "line")
                ?lineStyle list("dash" "dash")
                ?lineThickness list("thick" "thick")
                )
```

```
=> t
```



eyeBERLeft

```
eyeBERLeft(
     o_waveform
     n_start
     n_stop
     n_eyePeriod
     n_threshold
     n_noOfBins
     )
     => o_waveform / nil
```

Description

Returns a waveform representing the left-side bit error rate (BER) curve for the specified eye diagram.

Arguments

o_waveform	The eye diagram waveform.
n_start	Start time of the signal used to create the eye diagram.
n_stop	Stop time of the signal used to create the eye diagram.
n_eyePeriod	Eye period of the eye diagram.
n_threshold	Crossing threshold value on y axis.
n_noOfBins	Number of bins used to create histograms of crossing data.

Value Returned

o_waveform	Waveform representing the left-side bit error rate (BER) curve for the specified eye diagram.
`nil`	The specified eye diagram does not exist or the output waveform cannot be returned because of an error.

Examples

The following example creates an eye diagram from a `jitter` signal with time from `200n` to `400u` and an eye period of `80n`.

```
eye=eyeDiagram(v("jitter" ?result "tran-tran" ?resultsDir "/servers/user/design/
prbs.raw") 200n 400u 2*40n   ?autoCenter t )
```

```
=> srrWave:0x336321b0
```

The following example returns the waveform representing the left-side bit error rate (BER) curve for the specified eye diagram.

```
w1=eyeBERLeft(eye 200n 400u 80n 0.5 50)
=> srrWave:0x33632190
```

eyeBERLeftApprox

```
eyeBERLeftApprox(
     o_waveform
     n_start
     n_stop
     n_eyePeriod
     n_threshold
     n_noOfBins
     )
     => o_waveform / nil
```

Description

Returns a waveform representing the left-side bit error rate (BER) curve for the specified eye diagram beyond the output of `eyeBERLeft` by tail-fitting the left-side cross distribution.

Arguments

o_waveform	The eye diagram waveform.
n_start	Start time of the signal used to create the eye diagram.
n_stop	Stop time of the signal used to create the eye diagram.
n_eyePeriod	Eye period of the eye diagram.
n_threshold	Crossing threshold value on y axis.
n_noOfBins	Number of bins used to create histograms of crossing data.

Value Returned

o_waveform	Waveform representing the left-side bit error rate (BER) curve for the specified eye diagram.
`nil`	The specified eye diagram does not exist or the output waveform cannot be returned because of an error.

Examples

The following example creates an eye diagram from a `jitter` signal with time from `200n` to `400u` and an eye period of `80n`.

```
eye=eyeDiagram(v("jitter" ?result "tran-tran" ?resultsDir "/servers/user/design/
prbs.raw") 200n 400u 2*40n   ?autoCenter t )
```

```
=> srrWave:0x336321b0
```

The following example returns the waveform representing the left-side bit error rate (BER) curve for the specified eye diagram beyond the output of `eyeBERLeft` by tail-fitting the left-side cross distribution.

```
w1=eyeBERLeftApprox(eye 200n 400u 80n 0.5 50)
=> srrWave:0x33632420
```

eyeBERRight

```
eyeBERRight(
     o_waveform
     n_start
     n_stop
     n_eyePeriod
     n_threshold
     n_noOfBins
     )
     => o_waveform / nil
```

Description

Returns a waveform representing the right-side bit error rate (BER) curve for the specified eye diagram.

Arguments

o_waveform	The eye diagram waveform.
n_start	Start time of the signal used to create the eye diagram.
n_stop	Stop time of the signal used to create the eye diagram.
n_eyePeriod	Eye period of the eye diagram.
n_threshold	Crossing threshold value on y axis.
n_noOfBins	Number of bins used to create histograms of crossing data.

Value Returned

o_waveform	Waveform representing the right-side bit error rate (BER) curve for the specified eye diagram.
`nil`	The specified eye diagram does not exist or the output waveform cannot be returned because of an error.

Examples

The following example creates an eye diagram from a `jitter` signal with time from `200n` to `400u` and an eye period of `80n`.

```
eye=eyeDiagram(v("jitter" ?result "tran-tran" ?resultsDir "/servers/user/design/
prbs.raw") 200n 400u 2*40n   ?autoCenter t )
```

```
=> srrWave:0x336321b0
```

The following example returns the waveform representing the right-side bit error rate (BER) curve for the specified eye diagram.

```
w1=eyeBERRight(eye 200n 400u 80n 0.5 50)
=> srrWave:0x33632280
```

eyeBERRightApprox

```
eyeBERRightApprox(
     o_waveform
     n_start
     n_stop
     n_eyePeriod
     n_threshold
     n_noOfBins
     )
     => o_waveform / nil
```

Description

Returns a waveform representing the right-side bit error rate (BER) curve for the specified eye diagram beyond the output of `eyeBERRight` by tail-fitting the right-side cross distribution.

Arguments

o_waveform	The eye diagram waveform.
n_start	Start time of the signal used to create the eye diagram.
n_stop	Stop time of the signal used to create the eye diagram.
n_eyePeriod	Eye period of the eye diagram.
n_threshold	Crossing threshold value on y axis.
n_noOfBins	Number of bins used to create histograms of crossing data.

Value Returned

o_waveform	Waveform representing the right-side bit error rate (BER) curve for the specified eye diagram.
`nil`	The specified eye diagram does not exist or the output waveform cannot be returned because of an error.

Examples

The following example creates an eye diagram from a `jitter` signal with time from `200n` to `400u` and an eye period of `80n`.

```
eye=eyeDiagram(v("jitter" ?result "tran-tran" ?resultsDir "/servers/user/design/
prbs.raw") 200n 400u 2*40n   ?autoCenter t )
```

```
=> srrWave:0x336321b0
```

The following example returns the waveform representing the right-side bit error rate (BER) curve for the specified eye diagram beyond the output of `eyeBERRight` by tail-fitting the right-side cross distribution.

```
w1=eyeBERRightApprox(eye 200n 400u 80n 0.5 50)
=> srrWave:0x33632290
```

eyeHeightAtXY

```
eyeHeightAtXY(
     o_eyeDiagram
     f_x
     f_y
     [ ?output t_output ]
     )
     => f_eyeHight / 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 direction of y axis.

Note: The specified coordinates (x, y) must lie within the open eye whose height you want to calculate.

Arguments

o_eyeDiagram	Eye diagram whose height is to be calculated at the specified point.
f_x	X-axis coordinate of the points at which eye height is to be calculated.
f_y	Y-axis coordinate of the points at which eye height is to be calculated.
?output *t_output*	Specifies whether to calculate the total eye height or the eye height relative to the specified y-axis value. Valid values are: ■ `total`: Calculates the total eye height at the point (x, y). ■ `above`: Calculates the eye height that is above the specified y-axis value. ■ `below`: Calculates the eye height that is below the specified y-axis value. The default value is `total`.

Value Returned

f_eyeHeight	Eye height at the specified point inside the eye diagram.
nil	Eye height cannot be calculated because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/prbs.raw")
=> "/servers/user/design/prbs.raw"
```

The following example lists the results contained in the results directory `/servers/user/design/prbs.raw`.

```
results()
=> tran()
```

The following example selects the tran results.

```
selectResult('tran)
=> stdobj@0x321eab00
```

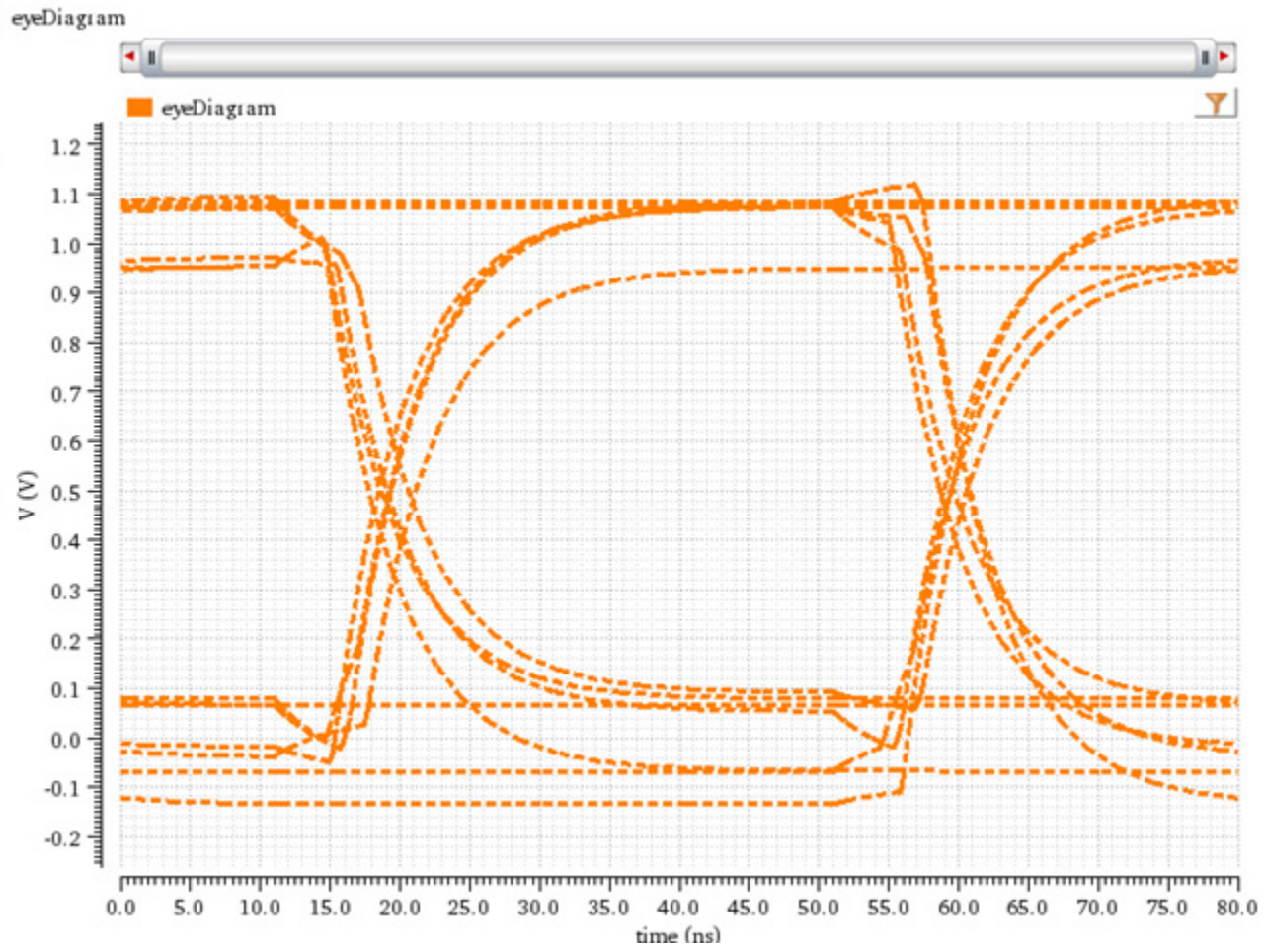
The following example creates a waveform object `eye` that represents an eye diagram created using a signal `jitter`, which is available in the `tran` results of the results directory `/servers/user/design/prbs.raw`.

```
eye=eyeDiagram(v("jitter" ?result "tran-tran") 200n 1.4u 2*40n   ?autoCenter t )
=> srrWave:0x364bc030
```

The following example plots the eye diagram in the Waveform window that you created using the function `awvCreatePlotWindow`.

```
awvPlotWaveform(
                window(3)
                list(eye)
                ?expr list("eyeDiagram")
                ?color list("y6")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("dot")
                ?lineThickness list("thick")
```

```
                    )
=> t
```



The following example places a horizontal marker at y=0.5V (500mV).

```
awvPlaceYMarker(window(3) 0.5 ?label "H1at500mV")
=> "horizMarker[1.1.1]"
```

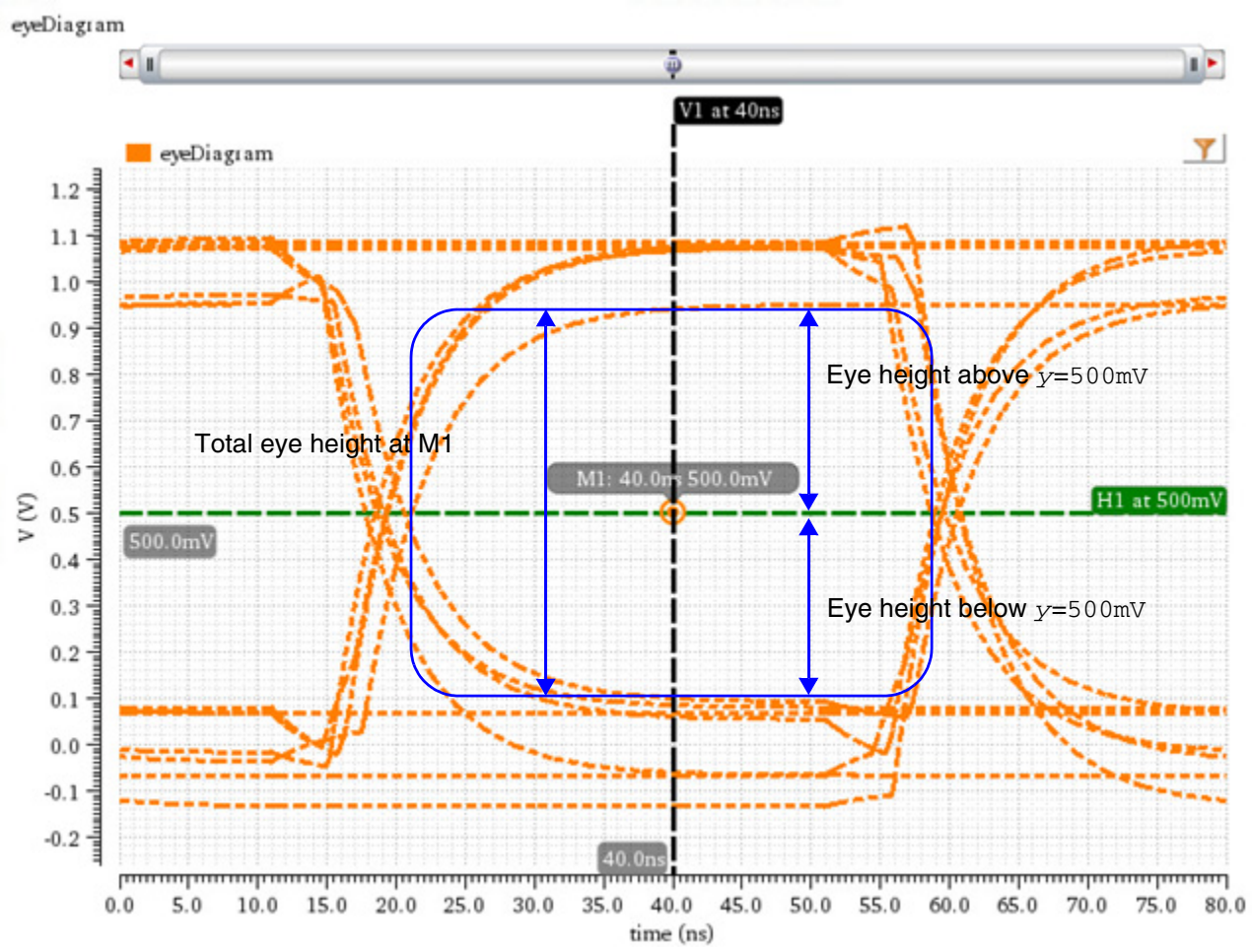
The following example places a vertical marker at x=40ns.

```
awvPlaceXMarker(window(3) 40n ?label "V1 at 40ns")
=> "vertMarker[1.1.1]"
```

The following example places a point marker M1 at x=40ns and y=500mV.

```
awvPlacePointMarker(window(3) 1 40n 500m ?positionMode "xy")
```

```
=> "pointMarker[1.1.1]"
```



The following example calculates the total eye height of eye diagram `eye` at point `M1` whose x-axis and y-axis coordinates are `40ns` and `0.5`, respectively.

```
eyeHeightAtXY(eye 40n 0.5 ?output "total")
=> 0.8399164
```

The following example calculates the eye height that is above y=`0.5V` (`500mV`) at point `M1` (x=`40ns`, y=`0.5`).

```
eyeHeightAtXY(eye 40n 0.5 ?output "above")
=> 0.4397585
```

The following example calculates the eye height that is below y=`0.5V` (`500mV`) at point `M1` (x=`40ns`, y=`0.5`).

```
eyeHeightAtXY(eye 40n 0.5 ?output "below")
=> 0.4001579
```

Note that total eye height at any point inside the eye diagram is equal to the sum of eye heights above and below that point.

$$eyeHeightAtXY_{total} = eyeHeightAtXY_{above} + eyeHeightAtXY_{below}$$

eyeMask

```
eyeMask(
     o_waveform
     t_xUnit
     [ @rest l_vertices ]
     )
     => o_waveform
```

Description

Creates a custom eye mask on an eye diagram at the specified vertices in the specified units.

Arguments

o_waveform	The eye diagram waveform on which eye mask is to be created.
t_xUnit	Units in which x coordinates of the vertices are specified. Valid values are: ■ `UI`: x coordinates are specified in unit interval. ■ `s`: x coordinates are specified in second.
`@rest` *l_vertices*	List specifying the coordinates of the eye mask. The coordinates must be ordered adjacently. You can also specify the coordinates of the eye mask using the `VAR` function.

Value Returned

o_waveform	Waveform ID representing eye diagram with the specified eye mask.

Examples

The following example creates a Waveform window and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/prbs.raw")
=> "/servers/user/design/prbs.raw"
```

The following example lists the results available in the currently open results directory.

```
results()
=> tran()
```

The following example selects the `tran` result of the results directory.

```
selectResult('tran)
=> stdobj@0x319b4b30
```

The following example creates a waveform object `jitter`, representing the waveform of the signal `jitter`, which is available in the `tran` result of the results directory.

```
jitter=v("jitter")
=> srrWave:0x35c81020
```

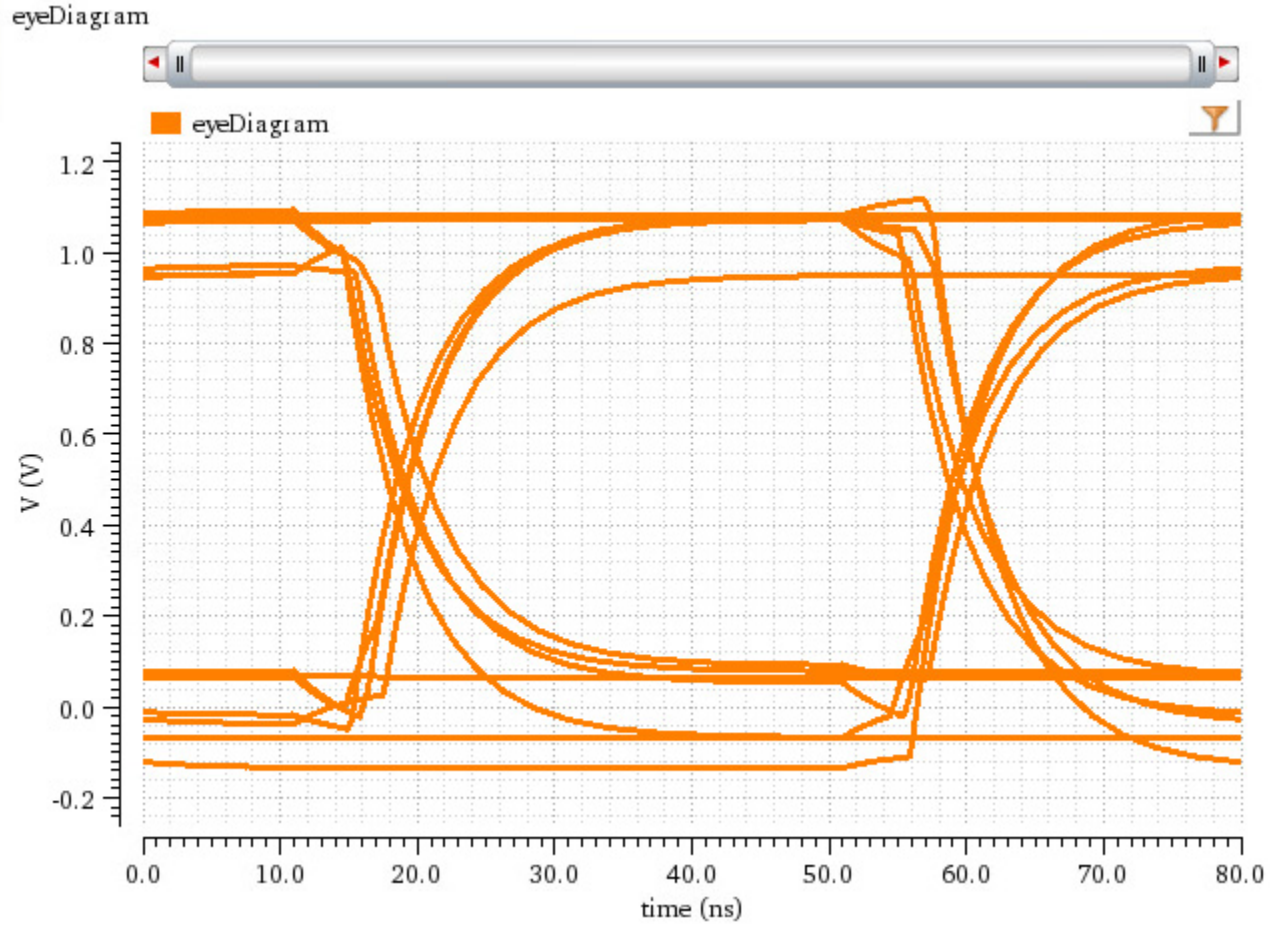
The following example creates a waveform object `eye`, representing an eye diagram that is generated by applying the `eyeDiagram` function on the `jitter` signal.

```
eye=eyeDiagram(jitter 200n 1.4u 2*40n ?autoCenter t)
=> srrWave:0x35c81030
```

The following example plots the waveform `eye` in the Waveform window that you created using the function `awvCreatePlotWindow`.

```
awvPlotWaveform(
                win1
                list(eye)
                ?expr list("eyeDiagram")
                ?color list("y6")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                )
```

```
=> t
```



The following example creates another Waveform window and returns its window ID.

```
win2=awvCreatePlotWindow()
=> window:4
```

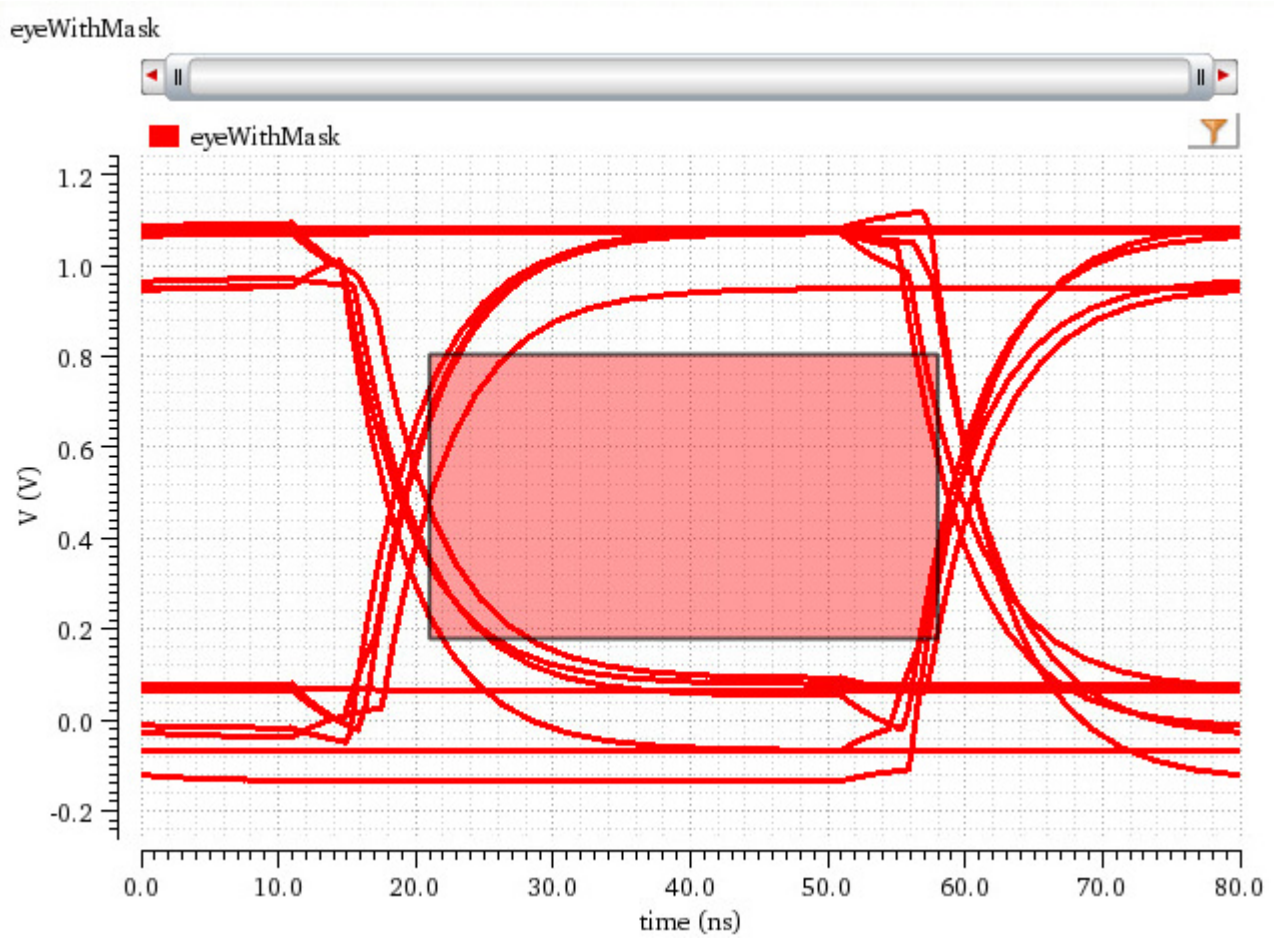
The following example creates a waveform object `eyeMaskR` that represents the eye diagram `eye` with an eye mask of the `rectangle` geometry placed over it at the specified coordinates.

```
eyeMaskR=eyeMask(eye "s" '(21n 0.179) '(21n 0.805) '(58n 0.805) '(58n 0.179))
=> srrWave:0x35c37040
```

The following example plots the waveform object `eyeMaskR`.

```
awvPlotWaveform(
                win2
                list(eyeMaskR)
                ?expr list("eyeWithMask")
```

```
                ?color list("y1")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                )
=> t
```



The following example creates another Waveform window and returns its window ID.

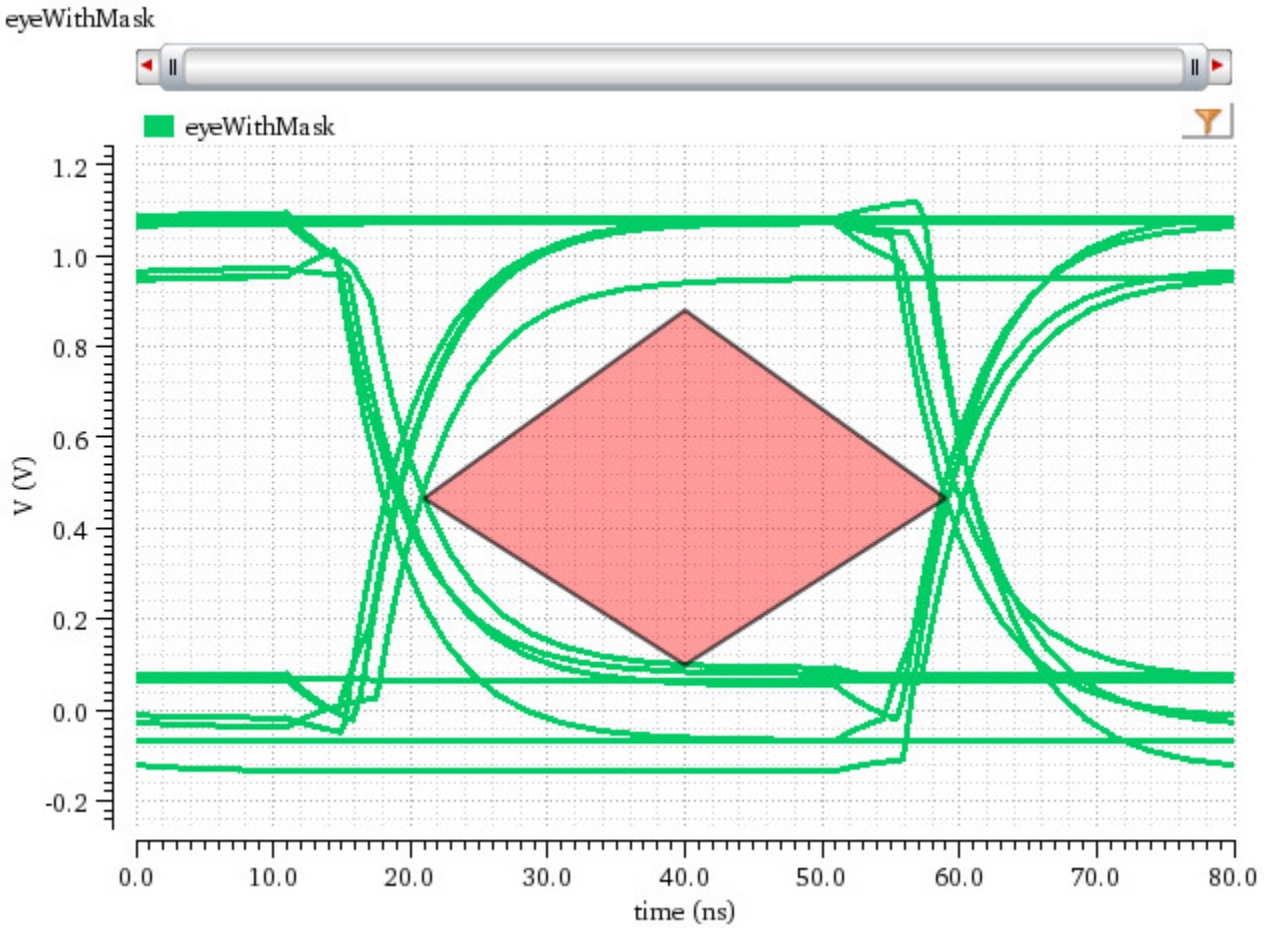
```
win3=awvCreatePlotWindow()
=> window:5
```

The following example creates a waveform object `eyeMaskD` that represents the eye diagram `eye` with an eye mask of the `diamond` geometry placed over it at the specified coordinates.

```
eyeMaskD=eyeMask(eye "s" '(21n 0.466) '(40n 0.880) '(59n 0.466) '(40n 0.099))
=> srrWave:0x35c37050
```

The following example plots the waveform object `eyeMaskD`.

```
awvPlotWaveform(
                win3
                list(eyeMaskD)
                ?expr list("eyeWithMask")
                ?color list("y2")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                )
=> t
```



The following example creates a Waveform window and returns its window ID.

```
win4=awvCreatePlotWindow()
=> window:6
```

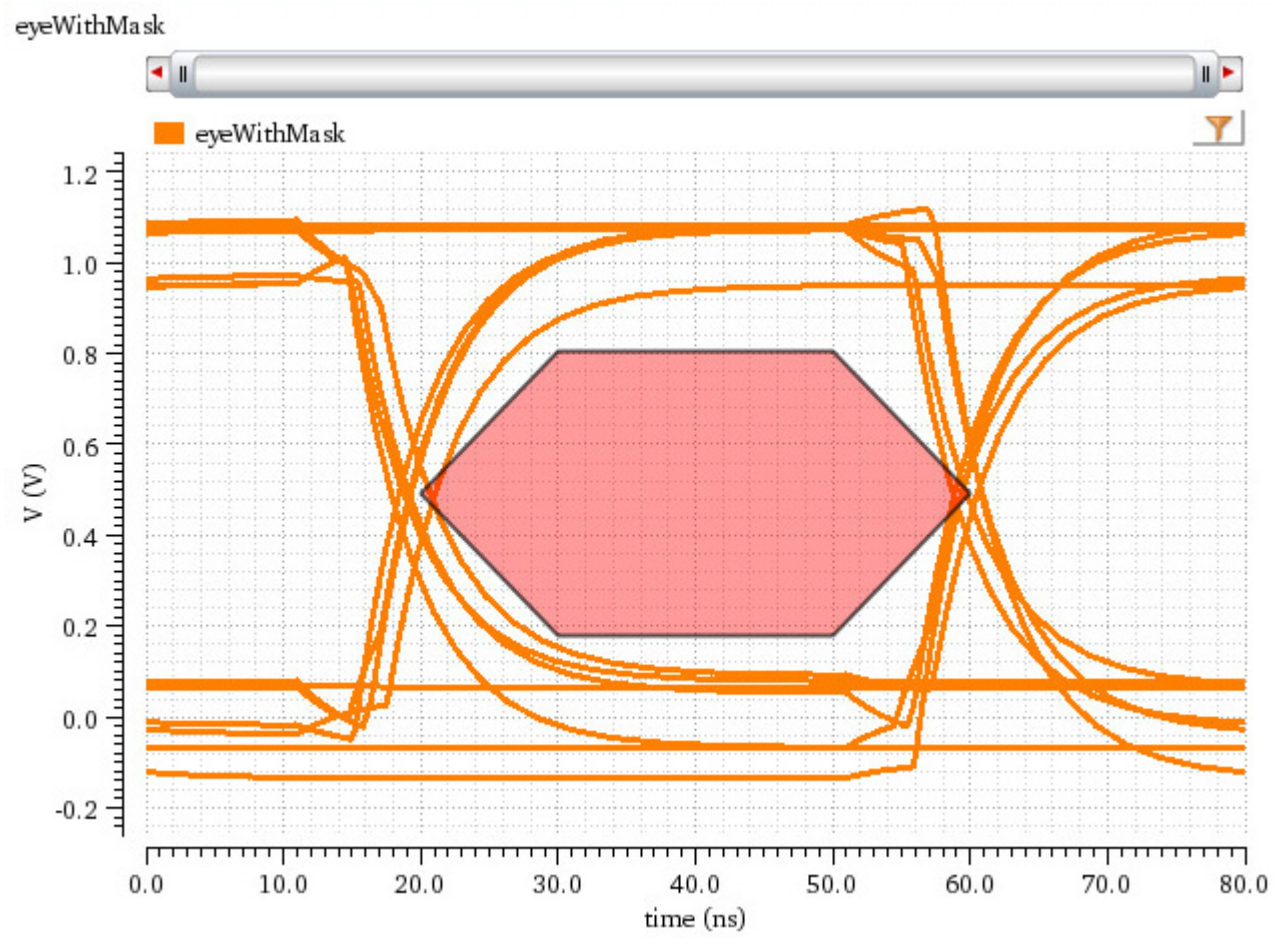
The following example creates a waveform object `eyeMaskH` that represents the eye diagram `eye` with an eye mask of the `hexagon` geometry placed over it at the specified coordinates.

```
eyeMaskH=eyeMask(eye "s" '(20n 0.492) '(30n 0.805) '(50n 0.805) '(60n 0.492) '(50n
0.179) '(30n 0.179))
=> srrWave:0x35c37060
```

The following example plots the waveform object `eyeMaskH`.

```
awvPlotWaveform(
                win4
                list(eyeMaskH)
                ?expr list("eyeWithMask")
                ?color list("y6")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                )
```

```
=> t
```



eyeMaskViolationPeriodCount

```
eyeMaskViolationPeriodCount(
     o_eyeMaskWaveform
     )
     => n_periods / nil
```

Description

Returns the number of periods that contain eye mask violation.

Arguments

o_eyeMaskWaveform

Waveform ID of the eye diagram with eye mask overlaid on it.

Value Returned

n_periods	Scalar value showing number of periods that contain an eye mask violation.
`nil`	If there is an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/prbs.raw")
=> "/servers/user/design/prbs.raw"
```

The following example lists the results available in the currently open results directory.

```
results()
=> tran()
```

The following example selects the `tran` result of the results directory.

```
selectResult('tran)
=> stdobj@0x31b6fb30
```

The following example creates a waveform object `jitter`, representing the waveform of the signal `jitter`, which is available in the `tran` result of the results directory.

```
jitter=v("jitter")
=> srrWave:0x35e3b020
```

The following example creates a waveform object `eye`, representing an eye diagram that is generated by applying the `eyeDiagram` function on the `jitter` signal.

```
eye=eyeDiagram(jitter 200n 1.4u 2*40n ?autoCenter t)
=> srrWave:0x35e3b030
```

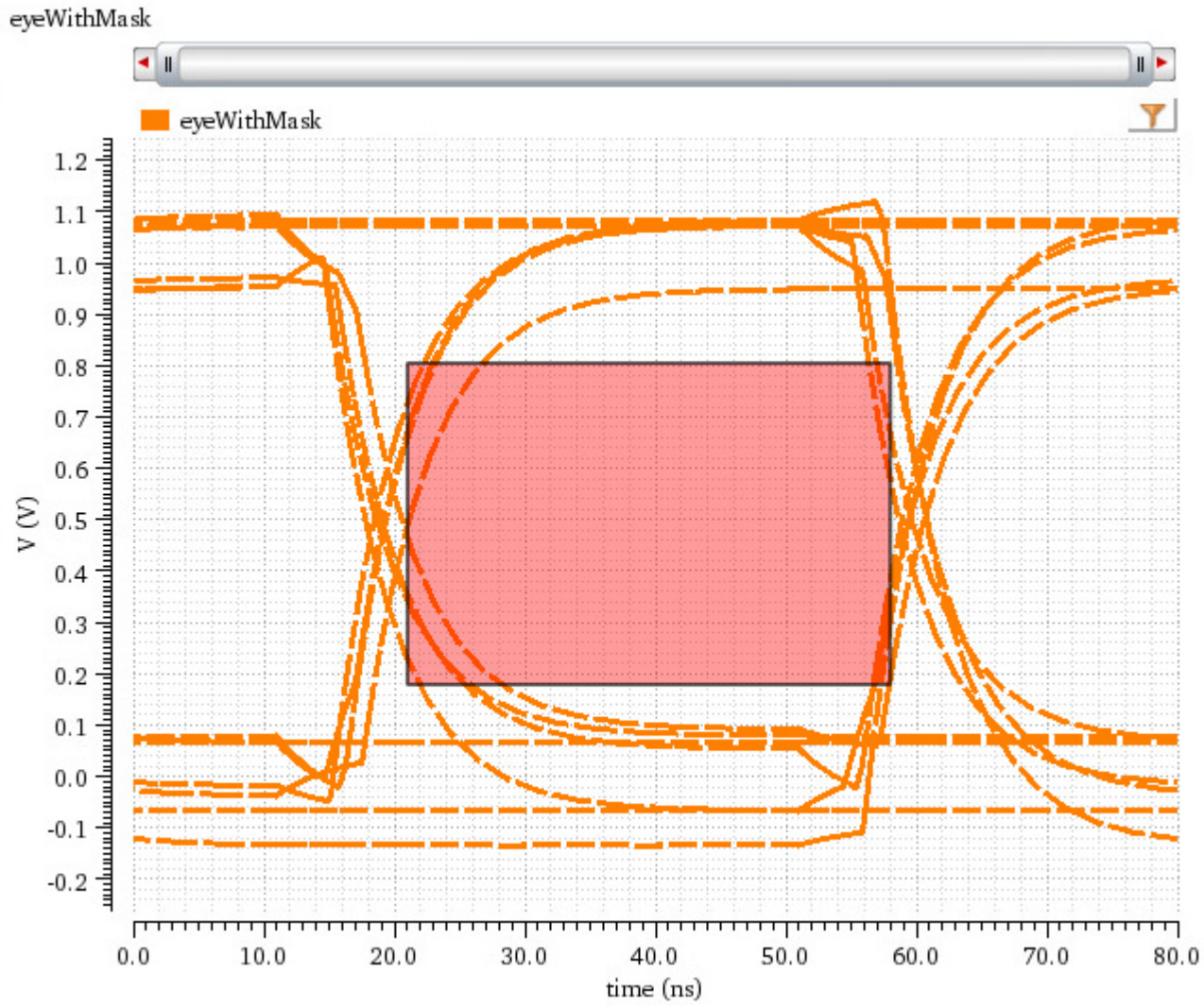
The following example creates a waveform object `eyeMaskR` that represents the eye diagram `eye` with an eye mask of the `rectangle` geometry placed over it at the specified coordinates.

```
eyeMaskR=eyeMask(eye "s" '(21n 0.179) '(21n 0.805) '(58n 0.805) '(58n 0.179))
=> srrWave:0x35df1040
```

The following example plots the waveform object `eyeMaskR` in the Waveform window that you created using the function `awvCreatePlotWindow`.

```
awvPlotWaveform(
                win1
                list(eyeMaskR)
                ?expr list("eyeWithMask")
                ?color list("y6")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                )
```

```
=> t
```



Observe that the eye mask appears in red color, which indicates that there are number of periods that contain eye mask violations.

The following example returns the number of periods for which the eye diagram intersects with the specified eye mask `eyeMaskR`.

```
eyeMaskViolationPeriodCount(eyeMaskR)
=> 12
```

The following example creates a Waveform window and returns its window ID.

```
win2=awvCreatePlotWindow()
```

```
=> window:4
```

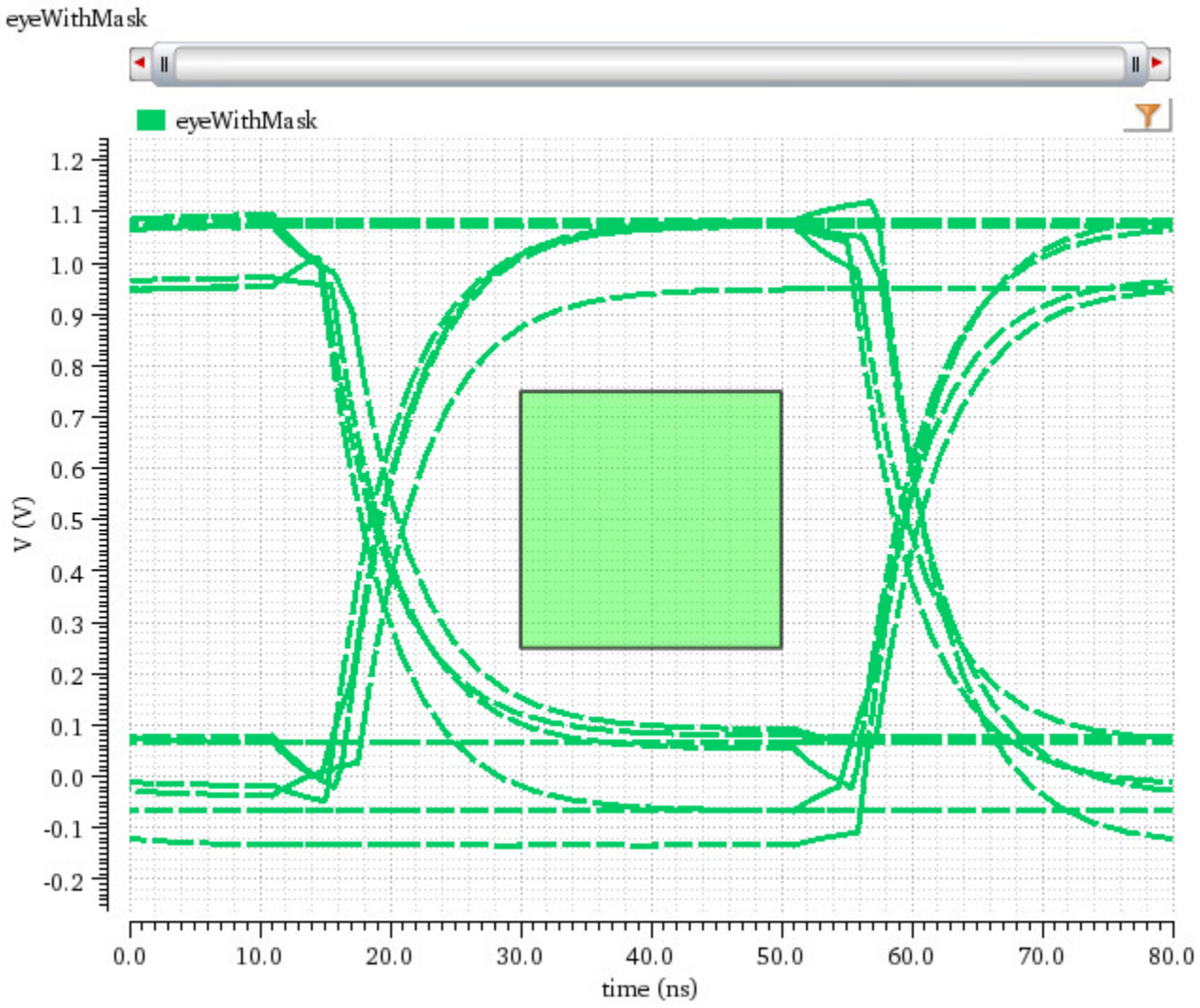
The following example creates another waveform object `eMaskR` that represents the eye diagram `eye` with an eye mask of the `rectangle` geometry placed over it at the specified coordinates.

```
eMaskR=eyeMask(eye "s" '(30n 0.250) '(30n 0.750) '(50n 0.750) '(50n 0.250))
=> srrWave:0x35df1050
```

The following example plots the waveform object `eMaskR` in the Waveform window that you created using the function `awvCreatePlotWindow`.

```
awvPlotWaveform(
                win2
                list(eMaskR)
                ?expr list("eyeWithMask")
                ?color list("y2")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                )
```

```
=> t
```



Observe that the eye mask appears in green color, which indicates that there are no periods that contain eye mask violations.

The following example returns the number of periods for which the eye diagram intersects with the specified eye mask `eMaskR`.

```
eyeMaskViolationPeriodCount(eMaskR)
=> 0
```

eyePeakToPeakJitter

```
eyePeakToPeakJitter(
     o_eyeDiagram
     f_threshold
     n_eyeStart
     n_eyeEnd
     )
     => x_p2pJitter / nil
```

Description

Calculates the peak-to-peak jitter at the specified threshold within the region of an eye diagram. The peak-to-peak jitter is the time between the first and the last crossing.

Arguments

o_eyeDiagram	Waveform object for the eye diagram
f_threshold	Crossing threshold at which the peak-to-peak jitter is calculated
n_eyeStart	Start time of the crossing
n_eyeEnd	End time of the crossing

Value Returned

x_p2pJitter	Peak-to-peak jitter at the specified threshold within the region of an eye diagram.
`nil`	Peak-to-peak jitter cannot be calculated because of an error.

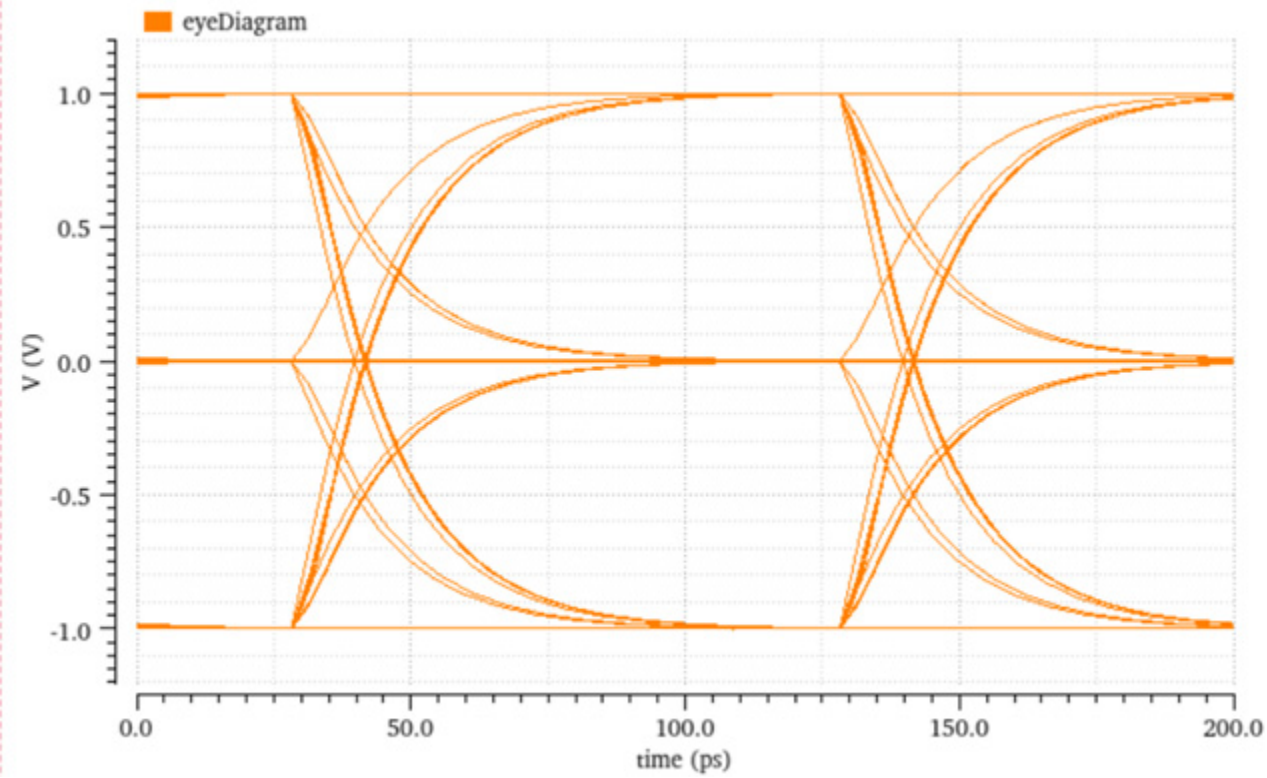
Examples

The following example calculates the peak-to-peak jitter at the threshold value `0` for the eye diagram represented by the waveform object, `eye`.

```
openResults("./simulation/PAM3_Eye_Diagram/pam3_testbench/maestro/results/
maestro/Interactive.13/psf/PAM3_Eye_Diagram_pam3_testbench_1/psf")
;Opens the simulation results stored in the specified directory.
=> "./simulation/PAM3_Eye_Diagram/pam3_testbench/maestro/results/maestro/
Interactive.13/psf/PAM3_Eye_Diagram_pam3_testbench_1/psf"


results()
```

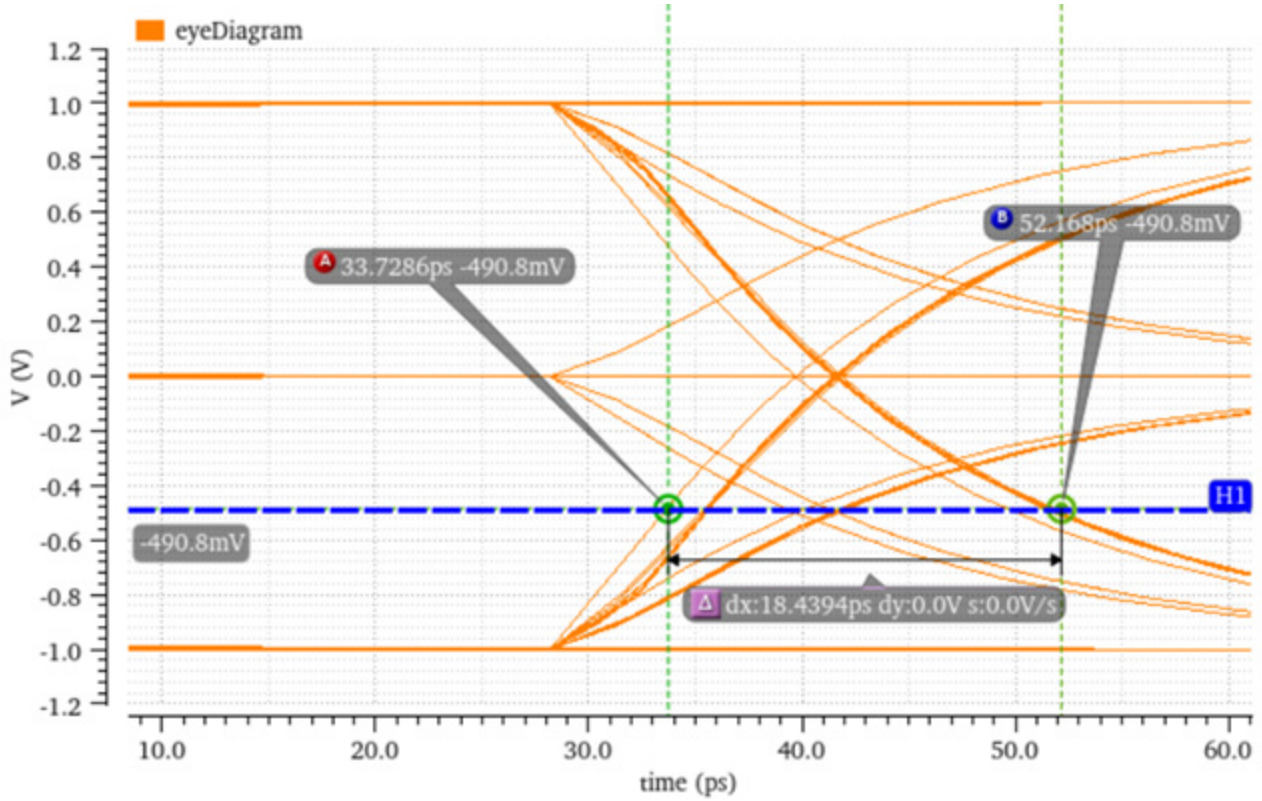
```
;Lists the results available in the specified results directory.
=> tran(tranOp model instance output designParamVals
    primitives subckts variables
)

selectResults('tran)
;Selects the result tran from the list of available results
=> stdobj@0x3945fc50

outputs();
;Lists the outputs available in the selected result tran.
=> ("/out1" "V0.p" "/in1")

Vout=v("/out1")
;Created a waveform object Vout representing the waveform of the voltage signal for
output out1.
=> srrWave:0x3e1ce020

awvCreatePlotWindow()
;Creates a Waveform window and returns its window ID
=> window:3

vvSetGraphBackground("#ffffff" awvGetCurrentWindow())
;Sets the background color of the current Waveform window to White.
=> (("graphWindow[1.1.1]")
    (("viva.graphFrame.background" "#ffffff"))
)

eye=leafValue( eyeDiagram(Vout 0.0 1e-07 (2 * 1e-10) ?triggerPeriod 1e-10
?autoCenter t) "trise1" "0.05*myUI" )
;Craetes a waveform object eye, representing the eye diagram created by applying
eyeDiagram function on the Vout signal.
=> srrWave:0x3e1ce030

awvPlotWaveform(
    window(3)
    list(eye)
    ?expr list("eyeDiagram")
    ?color list("y6")
    ?index list(1)
    )
;Plots the waveforms eye in the specified Waveform window.
```

```
=> t
```




```
eyePeakToPeakJitter(eye -490.8m 0 100p)
; Calculates the distance between the first (0p) and last crossing (100p)at the
threshold value -490.8mV of the eye diagram represented by the waveform object, eye.
=> 1.844226e-11
```

The calculated value of peak-to-peak jitter can be verified by placing the `A` and `B` markers on the first and last crossing at the threshold value of `-490.8mV`. The distance between these two markers is `18.4ps`.



eyeWidthAtXY

```
eyeWidthAtXY(
     o_eyeDiagram
     f_x
     f_y
     [ ?output t_output ]
     )
     => 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 direction of x axis.

Note: The specified coordinates (x, y) must lie within the open eye whose width you want to calculate.

Arguments

o_eyeDiagram	Eye diagram whose width is to be calculated at the specified point.
f_x	X-axis coordinate of the points at which eye width is to be calculated.
f_y	Y-axis coordinate of the points at which eye width is to be calculated.
?output *t_output*	Specifies whether to calculate the total eye width or the eye width relative to the specified x-axis value. Valid values are: ■ `total`: Calculates the total eye width at the point (x, y). ■ `left`: Calculates the eye width that is left to the specified x-axis value. ■ `right`: Calculates the eye width that is right to the specified x-axis value. The default value is `total`.

Value Returned

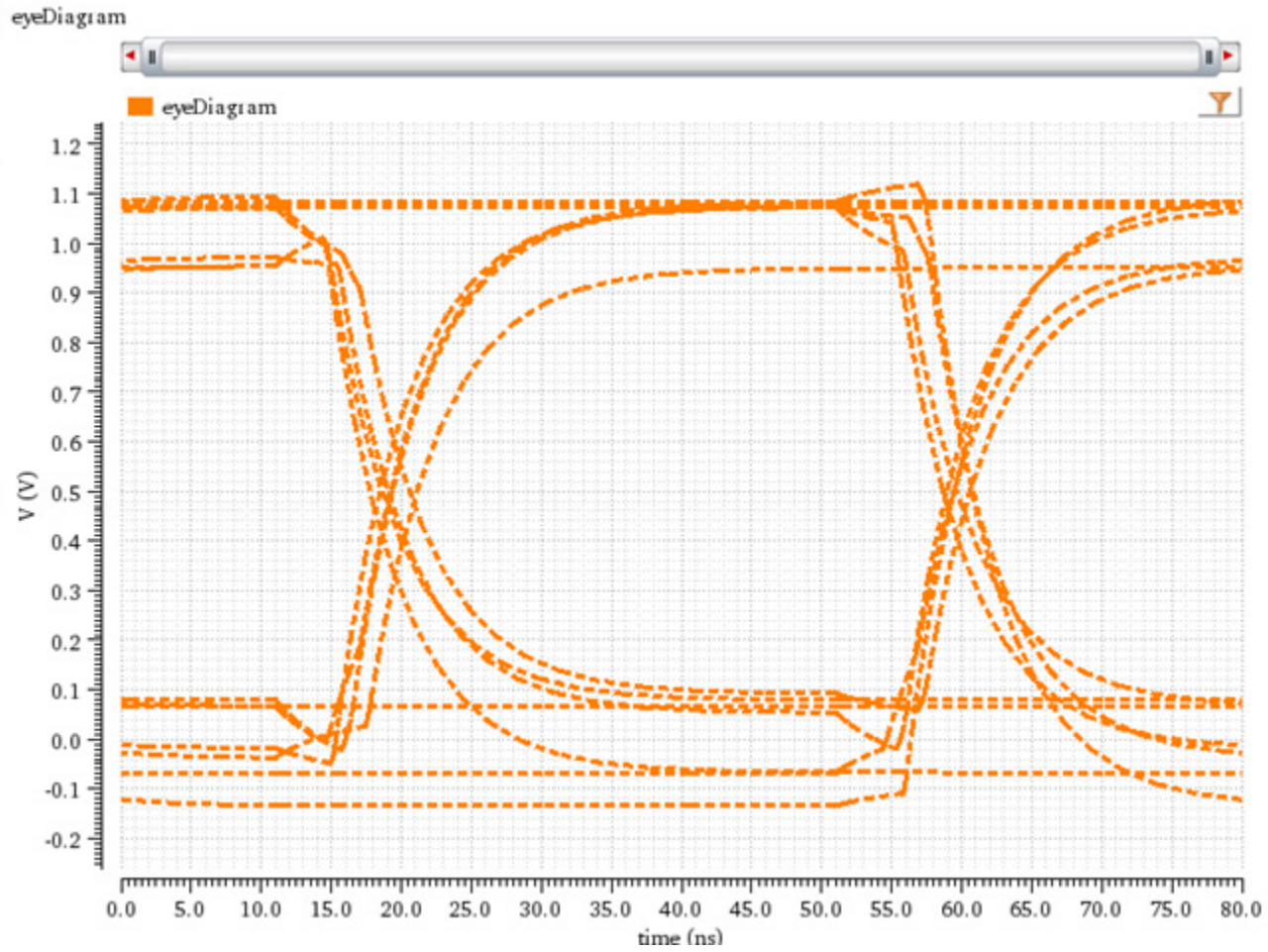
f_eyeWidth	Eye width at the specified point inside the eye diagram.
nil	Eye width cannot be calculated because of an error.

Examples

The following example calculates the eye width that is left to x=40ns at point M1 (x=40ns, y=0.5).

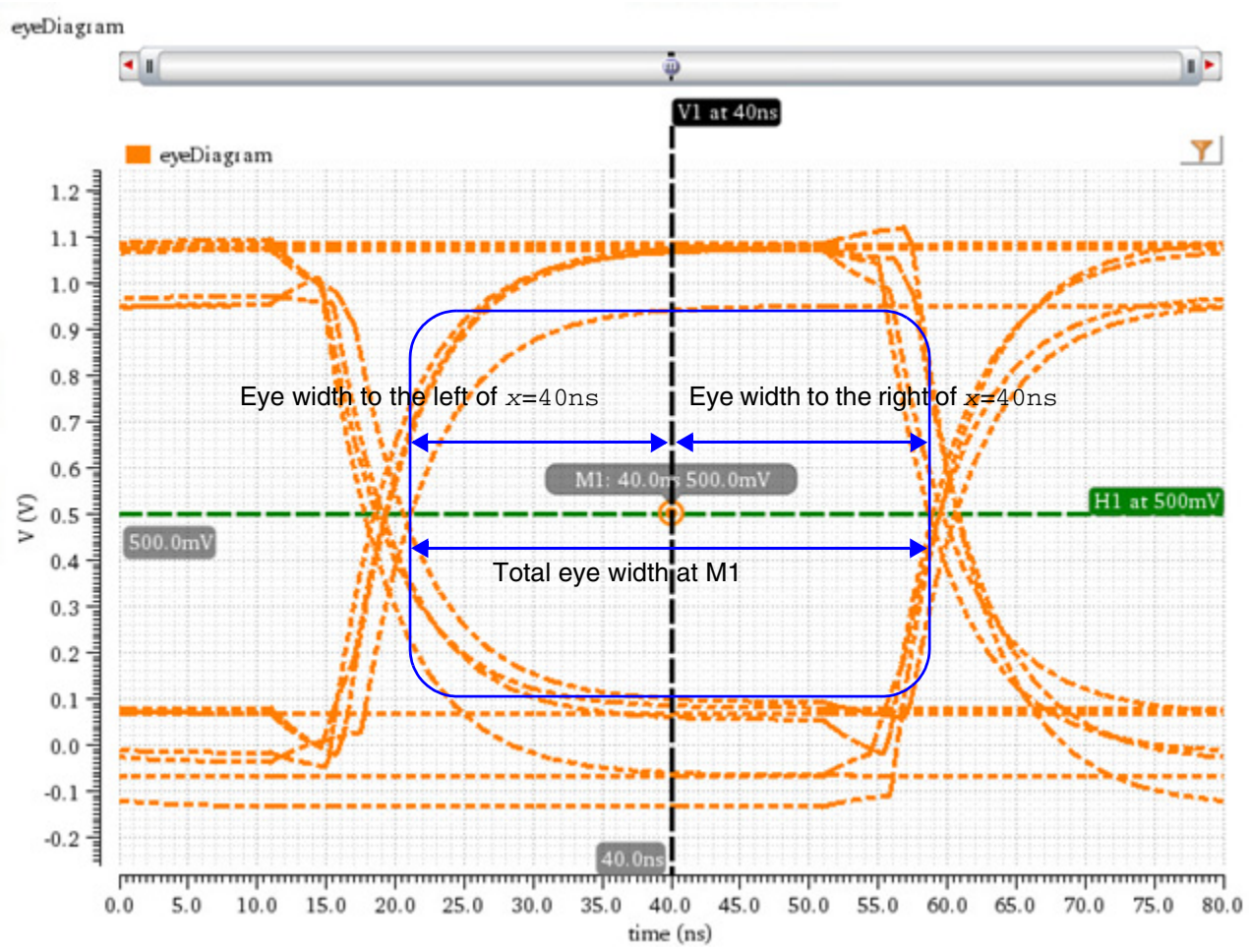
```
awvCreatePlotWindow()
;Creates a Waveform window and returns its window ID.
=> window:3

openResults("/servers/user/design/prbs.raw")
;Opens simulation results stored in the specified directory.
=> "/servers/user/design/prbs.raw"

results()
;Lists the results contained in the results directory /servers/user/design/
prbs.raw.
=> tran()

selectResult('tran)
;Selects the tran results.
=> stdobj@0x321eab00

eye=eyeDiagram(v("jitter" ?result "tran-tran") 200n 1.4u 2*40n   ?autoCenter t )
; Creates a waveform object eye that represents an eye diagram created using a
signal jitter, which is available in the tran results of the results directory /
servers/user/design/prbs.raw.
=> srrWave:0x364bc030

awvPlotWaveform(
                window(3)
                list(eye)
                ?expr list("eyeDiagram")
                ?color list("y6")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("dot")
                ?lineThickness list("thick")
```

```
                    )
;Plots the eye diagram in the Waveform window that you created using the function
awvCreatePlotWindow.
=> t
```




```
awvPlaceYMarker(window(3) 0.5 ?label "H1at500mV")
; Places a horizontal marker at y=0.5V (500mV).
=> "horizMarker[1.1.1]"

awvPlaceXMarker(window(3) 40n ?label "V1 at 40ns")
;Places a vertical marker at x=40ns.
=> "vertMarker[1.1.1]"

awvPlacePointMarker(window(3) 1 40n 500m ?positionMode "xy")
;Places a point marker M1 at x=40ns and y=500mV.
```

```
=> "pointMarker[1.1.1]"
```




```
eyeWidthAtXY(eye 40n 0.5 ?output "total")
; Calculates the total eye width of eye diagram eye at point M1 whose x-axis and
y-axis coordinates are 40ns and 0.5, respectively.
=> 3.754902e-08

eyeWidthAtXY(eye 40n 0.5 ?output "right")
; Calculates the eye width that is right to x=40ns at point M1 (x=40ns, y=0.5).
=> 1.868346e-08

eyeWidthAtXY(eye 40n 0.5 ?output "left")
;Calculates the eye width that is left to x=40ns at point M1 (x=40ns, y=0.5).

=> 1.886556e-08
```

Note that total eye width at any point inside the eye diagram is equal to the sum of eye widths to the left and to the right of that point.

$$eyeWidthAtXY_{total} = eyeWidthAtXY_{left} + eyeWiidthAtXY_{right}$$

famEval

```
famEval(
     l_expression
     [ t_sweepVariable1 ]
     [ s_sweepValue1 ]
     )
     => o_waveform / n_value / nil
```

Description

Evaluates the specified expression with the specified sweep variables.

Arguments

l_expression	Expression to be evaluated.
t_sweepVariable1	Name of the sweep variable. This is an optional argument.
s_sweepValue1	Value of the sweep variable. This is an optional argument.

Value Returned

o_waveform	Output waveform object.
n_value	Output scalar value.
nil	because of an error.

Examples

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/sweeptran.raw")
=> "/servers/user/design/sweeptran.raw"
```

The following example lists the results stored in the current results directory.

```
results()
=> tran()
```

The following example selects the `tran` result in the current results directory.

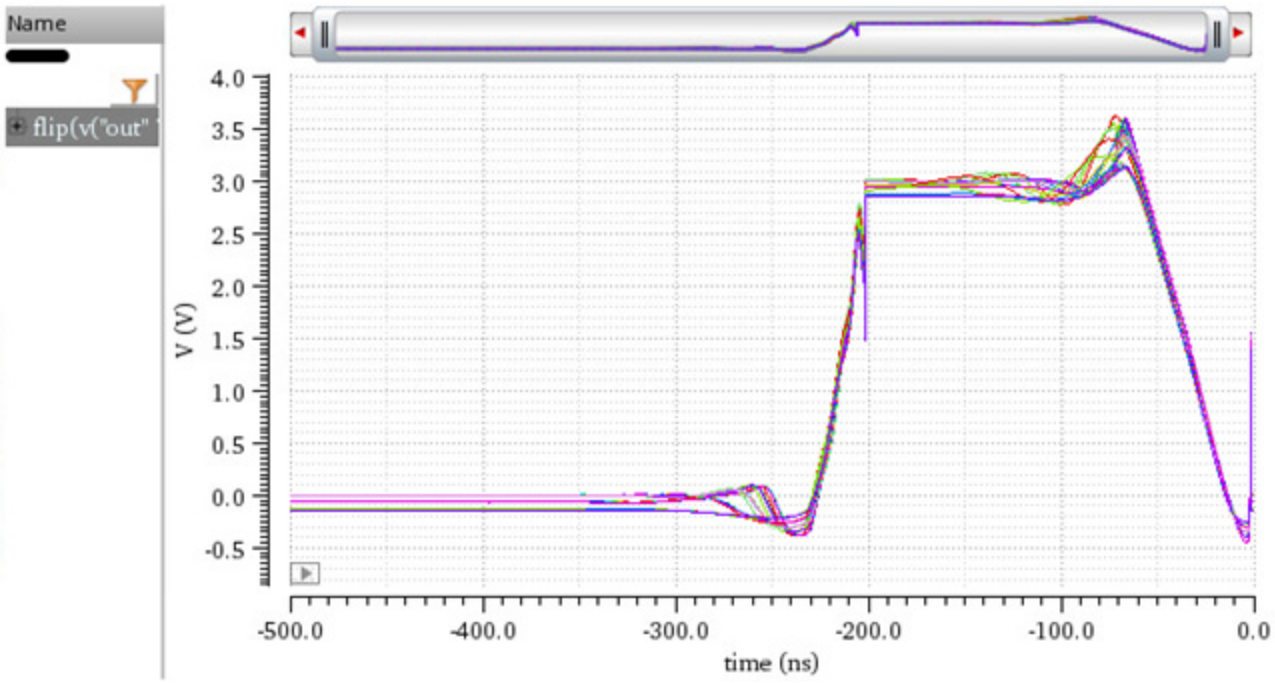
```
selectResults('tran)
=> stdobj@0x31f1f2f0
```

The following example creates a waveform object by evaluating the `flip` expression applied on the signal `out`, which is available in the `tran` result of the results directory `/servers/user/design/sweeptran.raw`.

```
familyWave=famEval(flip(v("out")))
=> srrWave:0x360da030
```

The following example plots the waveform object represented by `familyWave`.

```
plot familyWave
=> t
```



firstVal

```
firstVal(
     o_waveform
     )
     => n_value / nil
```

Description

Returns the first value from where the specified waveform starts on x axis.

Arguments

o_waveform	Waveform ID.

Value Returned

n_value	The first value from where the specified waveform starts from x axis

Examples

The following example returns the waveform ID of the eye diagram `eye` created from the jitter signal.

```
eye=eyeDiagram(v("jitter" ?result "tran-tran" ?resultsDir "/servers/user/design/
prbs.raw") 200n 400u 2*40n   ?autoCenter t )
=> srrWave:0x336321f0
```

The following example returns the waveform ID of the curve representing the left-side bit error rate (BER) curve for the eye diagram `eye`.

```
waveform=eyeBERLeft(eye 200n 400u 80n 0.5 100)
=> srrWave:0x336322e0
```

The following example returns the x-axis value from where the waveform returned by the function `eyeBERLeft` starts.

```
firstVal(waveform)
=>1.660711e-08
```

Indicates that the `waveform` starts from `x=16.60711ns`.

kurtosis

```
kurtosis(
     o_waveform
     )
     => n_kurtosis / nil
```

Description

Calculates the kurtosis of the specified waveform.

Kurtosis is a measurement of whether the data is heavy-tailed or light-tailed relative to a normal distribution.

Datasets with high kurtosis tend to have heavy tails or outliers. Datasets with low kurtosis tend to have light tails or few outliers.

Arguments

o_waveform	Waveform whose kurtosis is to be calculated.

Value Returned

n_kurtosis	Kurtosis value of the specified waveform.
`nil`	Kurtosis value cannot be calculated because of an error.

Examples

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/distribution.psf")
=> "/servers/user/distribution.psf"
```

The following example lists the results contained in the results directory.

```
results()
=> ("tran")
```

The following example selects the `tran` results of the results directory.

```
selectResults('tran)
=> stdobj@0x2a9de759
```

The following example creates a waveform object for the signal `v1net` stored in the `tran` results.

```
waveform=v("v1net" ?result "tran")
=> srrWave:0x3232c020
```

The following example calculates the kurtosis value of the specified waveform.

```
kurtosis(waveform)
=> 1.939457
```

lastVal

```
lastVal(
     o_waveform
     )
     => n_value / nil
```

Description

Returns the last value at which the specified waveform ends on x axis.

Arguments

o_waveform	Waveform ID.

Value Returned

n_value	The last value at which the specified waveform ends on x axis

Examples

The following example returns the waveform ID of the eye diagram `eye` created from the `jitter` signal.

```
eye=eyeDiagram(v("jitter" ?result "tran-tran" ?resultsDir "/servers/user/design/
prbs.raw") 200n 400u 2*40n   ?autoCenter t )
=> srrWave:0x336321f0
```

The following example returns the waveform ID of the curve representing the left-side bit error rate (BER) curve for the eye diagram `eye`.

```
waveform=eyeBERLeft(eye 200n 400u 80n 0.5 100)
=> srrWave:0x336322e0
```

The following example returns the last value at which the waveform returned by the function `eyeBERLeft` ends on x axis.

```
lastVal(waveform)
=> 2.353111e-08
```

Indicates that the `waveform` ends at `x=23.53111ns`.

leafValue

```
leafValue(
     o_waveform
     @rest l_args
     )
     => o_waveform / n_value / nil
```

Description

Returns the leaf waveform or the leaf value from the specified family of waveforms.

This function can also be used as a wrapper around the `value` function. It simplifies the syntax of the `value` function when there are multiple sweep variables.

This function is automatically applied when a leaf from a family of waveforms is sent to Calculator. It is automatically removed when expressions are sent from Calculator to the ADE outputs to prevent syntax errors.

Arguments

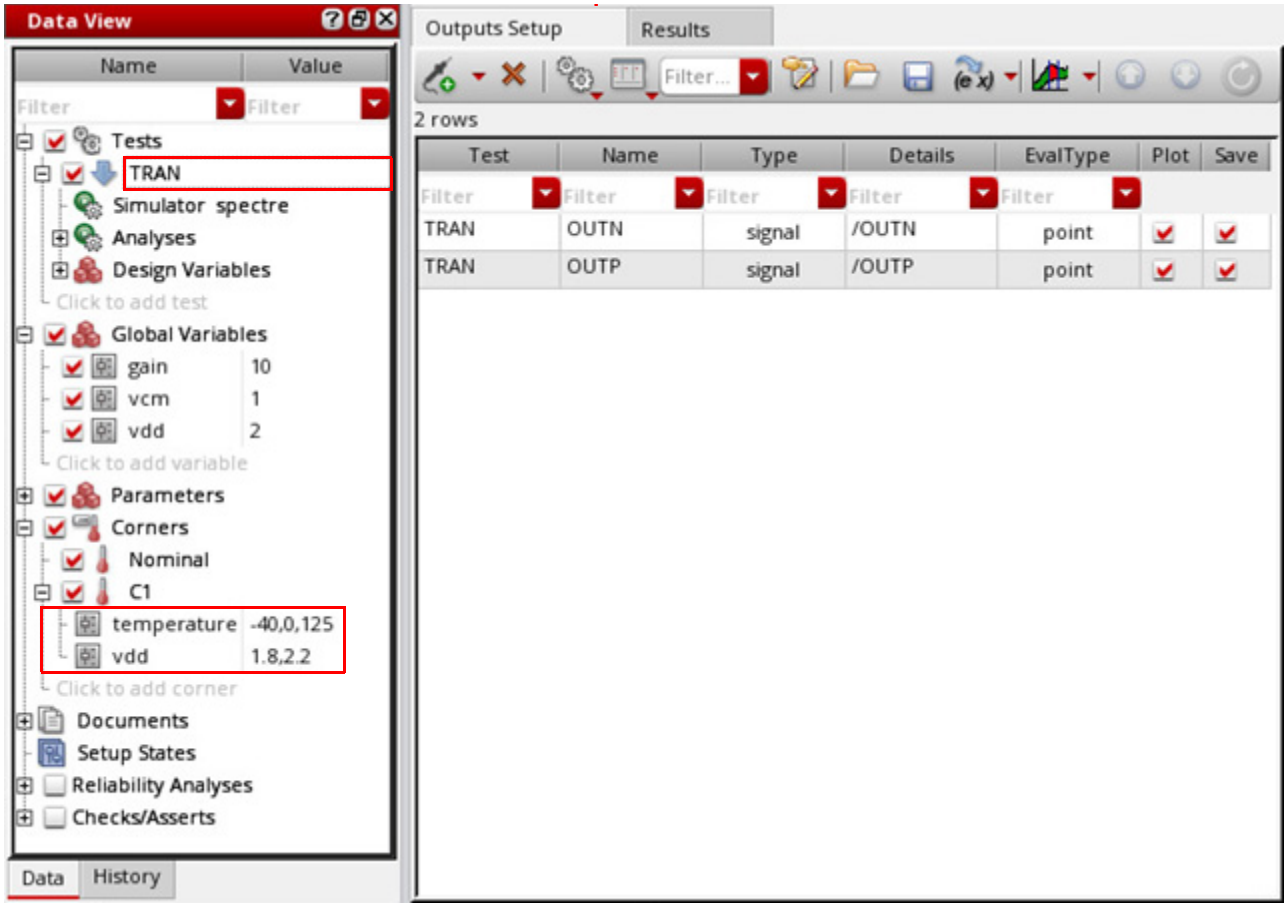
o_waveform	Family of waveforms.
`@rest` *l_args*	List of additional arguments that can be passed to the function. The list contains the name-value pairs of sweep variables specifying the leaf to be selected from the family of waveforms. *t_sweepVariable1Name g_sweepVariable1Value t_sweepVariable2Name g_sweepVariable2Value* ... For example: `"vdd" 1.8 "temperature" -40`

Value Returned

o_waveform	Leaf waveform.
n_value	Leaf scalar value.
`nil`	Leaf waveform or leaf value cannot be returned because of an error.

Examples

Consider the following test setup in ADE Assembler, where the test `TRAN` has two corners `Nominal` and `C1`. The corner `C1` contains sweeping variables, `temperature` and `vdd`.



After running the simulation, you can use the following command to open simulation results stored in the specified results directory.

```
openResults("/servers/user/testCase/simulation/lib/cell/maestro/results/maestro/
Interactive.2/psf/TRAN/psf")
=> "/servers/user/testCase/simulation/lib/cell/maestro/results/maestro/
Interactive.2/psf/TRAN/psf"
```

The following example lists the results stored in the current results directory.

```
results()
=> tran(tranOp designParamVals variables)
```

The following example selects the result `tran`.

```
selectResults('tran)
=> stdobj@0x32ac6e78
```

The following example lists the outputs stored in the selected result `tran`.

```
outputs()
=>
( "/OUTN" "/OUTP")
```

The following example creates a family of transient waveforms for the specified net `OUTP`.

```
famOUTP=VT("/OUTP")
=> srrWave:0x34df5020
```

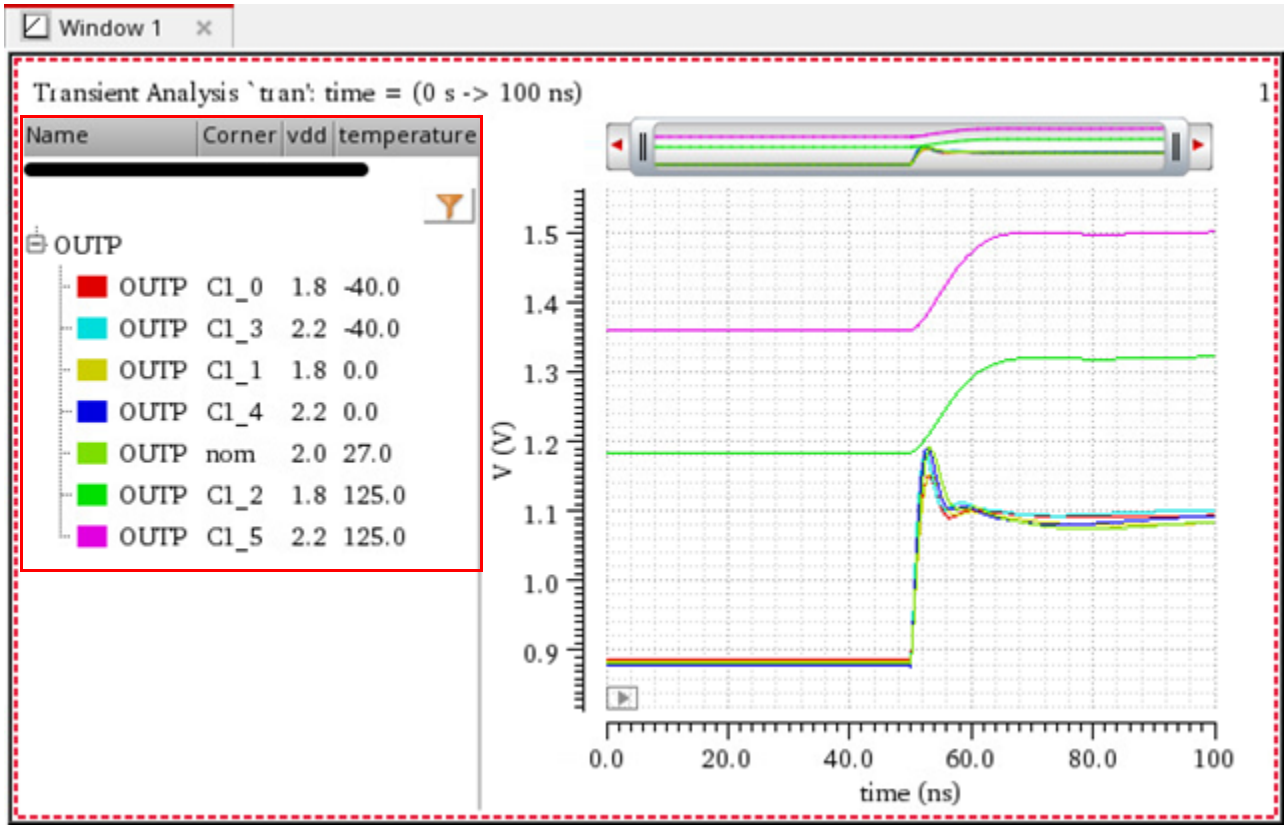
The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example plots the waveform object `famOUTP` in the Waveform window `win1`.

```
awvPlotWaveform(
                win1
                list(famOUTP)
                ?expr list("OUTP")
)
=> t
```

Note that `OUTP` contains `7` child traces that are plotted for corners, `Nominal` and `C1` and different values of sweep variables `vdd` and `temperature`.



You can use the function `leafValue` to individually create these `7` leaves from a family of waveforms `OUTP`.

The following examples create `7` waveform objects, each representing an individual leaf for the specified values of the sweep variables `vdd` and `temperature`.

```
leaf1=leafValue(VT("/OUTP") "vdd" 1.8 "temperature" -40)
=> srrWave:0x34df50e0
leaf2=leafValue(VT("/OUTP") "vdd" 2.2 "temperature" -40)
=> srrWave:0x34df5100
leaf3=leafValue(VT("/OUTP") "vdd" 1.8 "temperature" 0)
=> srrWave:0x34df5120
leaf4=leafValue(VT("/OUTP") "vdd" 2.2 "temperature" 0)
=> srrWave:0x34df5140
leaf5=leafValue(VT("/OUTP") "vdd" 2.0 "temperature" 27)
=> srrWave:0x34df5160
leaf6=leafValue(VT("/OUTP") "vdd" 1.8 "temperature" 125)
```

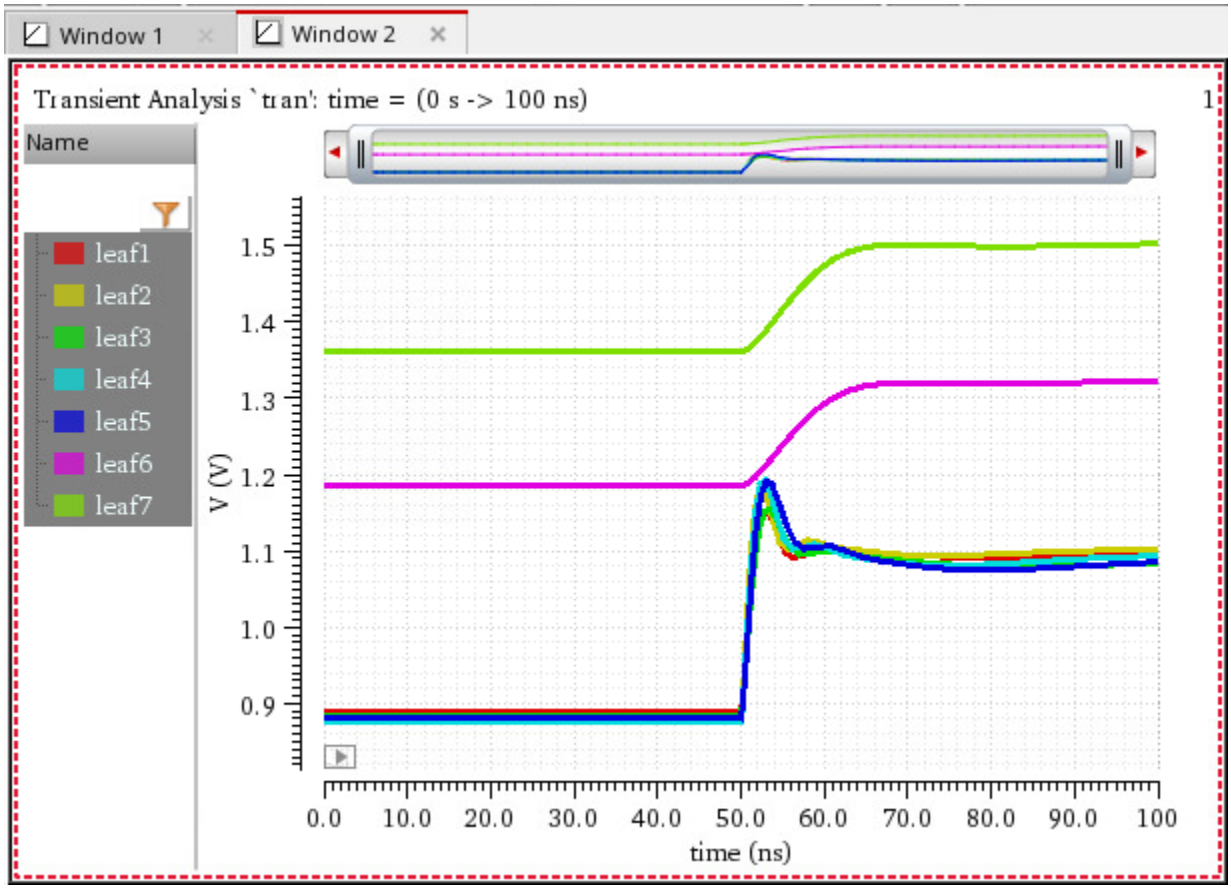
```
=> srrWave:0x34df5180
leaf7=leafValue(VT("/OUTP") "vdd" 2.2 "temperature" 125)
=> srrWave:0x34df51a0
```

The following example creates another Waveform window and returns its window ID.

```
win2=awvCreatePlotWindow()
=> window:4
```

The following example plots these leaves in the Waveform window `win2`.

```
awvPlotWaveform(
                win2
                list(leaf1 leaf2 leaf3 leaf4 leaf5 leaf6 leaf7)
                ?expr list("leaf1" "leaf2" "leaf3" "leaf4" "leaf5" "leaf6"
"leaf7")
)
=> t
```



mu

```
mu(
     o_s11Waveform
     o_s12Waveform
     o_s21Waveform
     o_s22Waveform
     )
     => 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_s11Waveform	Waveform object representing the S-Parameter `s11`.
o_s12Waveform	Waveform object representing the S-Parameter `s12`.
o_s21Waveform	Waveform object representing the S-Parameter `s21`.
o_s22Waveform	Waveform object representing the S-Parameter `s22`.

Value Returned

o_waveform	Waveform object indicating the minimum distance between the origin of the unit Smith chart and the load unstable region.
`nil`	Waveform object cannot be created because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results of S-Parameters analysis stored in the specified results directory.

```
openResults("/servers/user/MU/simulation/ampTest/spectre/schematic/psf")
```

```
=> "/servers/user/MU/simulation/ampTest/spectre/schematic/psf"
```

The following example returns the waveform object representing S-Parameter `s11`.

```
s11=aaSP(1 1)
=> srrWave:0x37627020
```

The following example returns the waveform object representing S-Parameter `s12`.

```
s12=aaSP(1 2)
=> srrWave:0x37627030
```

The following example returns the waveform object representing S-Parameter `s21`.

```
s21=aaSP(2 1)
=> srrWave:0x37627040
```

The following example returns the waveform object representing S-Parameter `s22`.

```
s22=aaSP(2 2)
=> srrWave:0x37627050
```

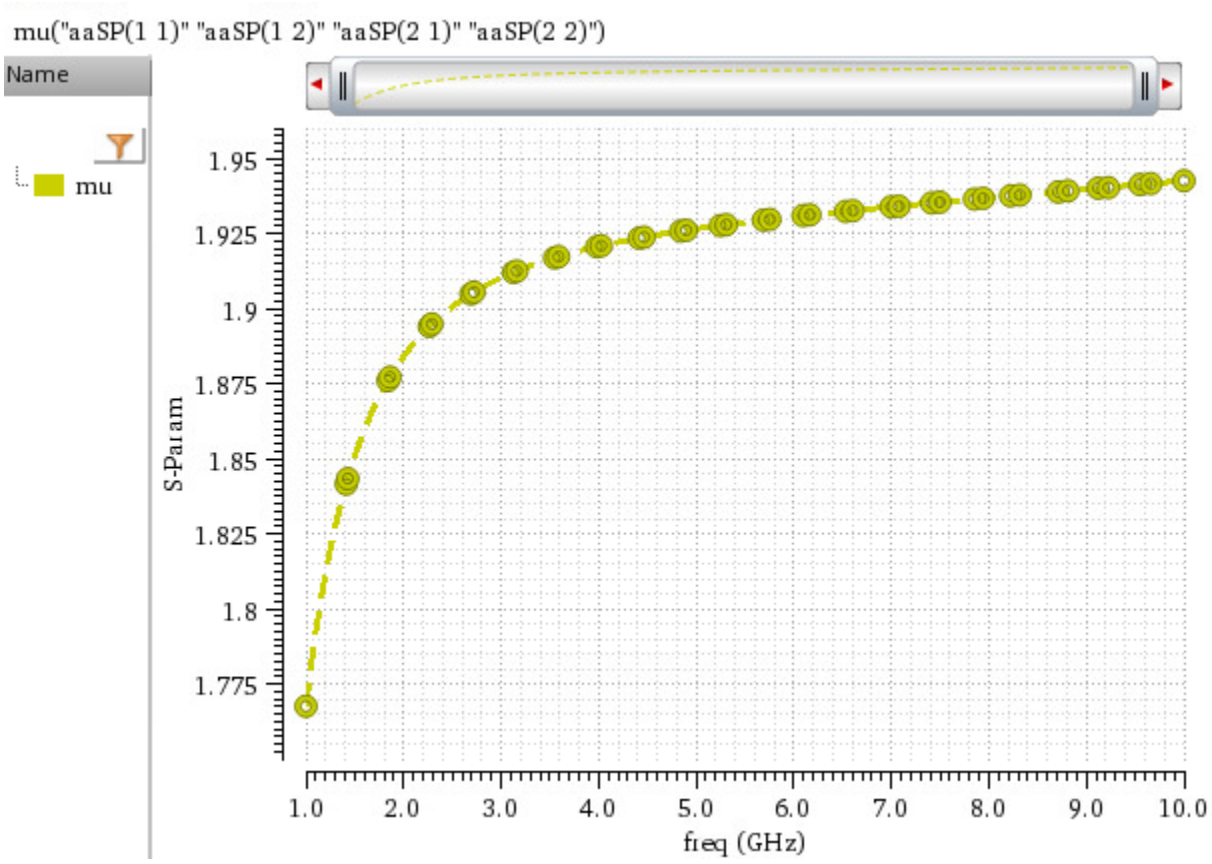
The following example creates a waveform object `muWaveform`, representing the minimum distance between the origin of the unit Smith chart and the load unstable region.

```
muWaveform=mu(s11 s12 s21 s22)
=> srrWave:0x37627130
```

The following example plots the waveform object `muWaveform`.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(muWaveform)
                ?expr list("mu")
                ?color list("y8")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("o")
                )
```

```
=> t
```



Mu

```
Mu(
     t_resultsDirectory
     )
     => 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

t_resultsDirectory

Path to the results directory that contains results of S-Parameter analysis.

Value Returned

o_waveform	Waveform object represnting the minimum distance between the origin of the unit Smith chart and the load unstable region.
`nil`	Waveform object cannot be created because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results of S-Parameters analysis stored in the specified results directory.

```
openResults("/servers/user/MU/simulation/ampTest/spectre/schematic/psf")
=> "/servers/user/MU/simulation/ampTest/spectre/schematic/psf"
```

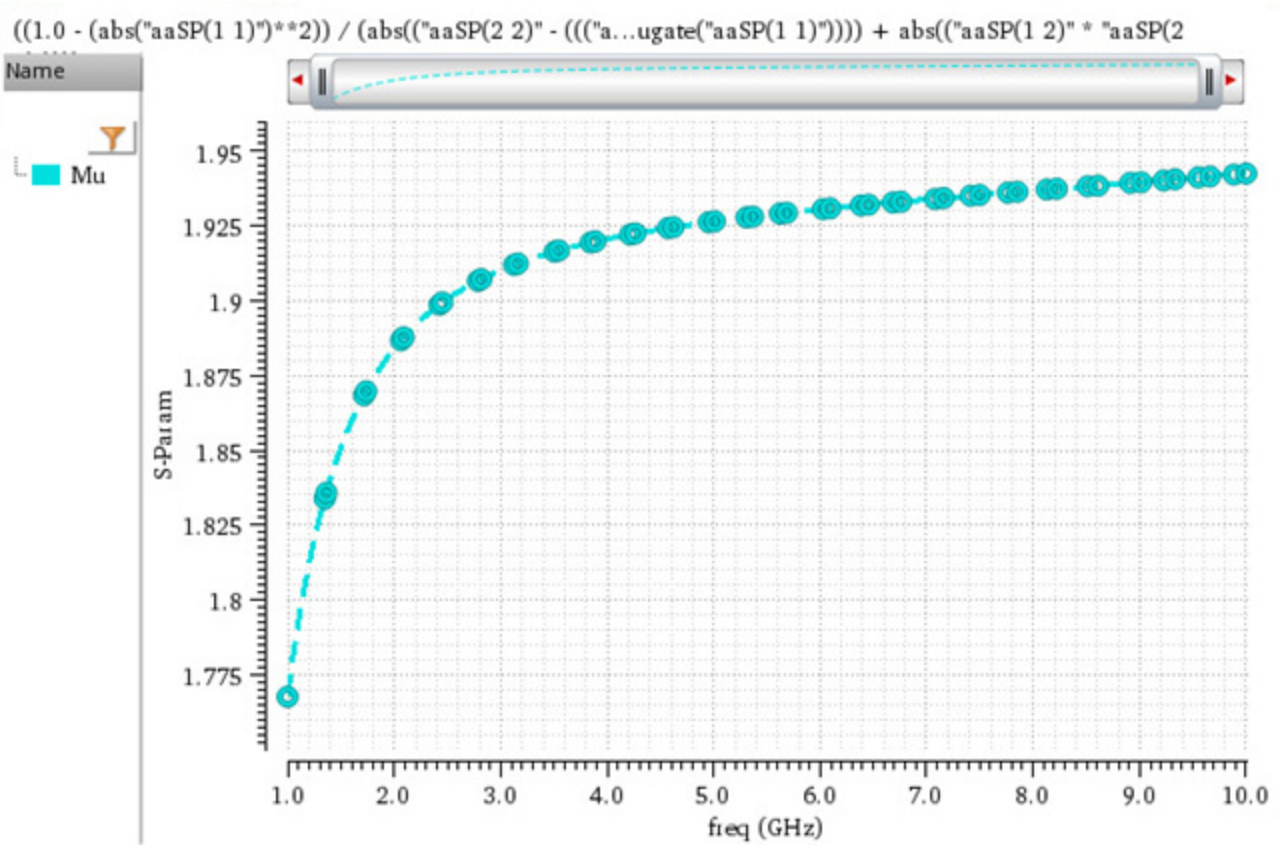
The following example creates a waveform object `MuWaveform`, representing the minimum distance between the origin of the unit Smith chart and the load unstable region.

```
MuWaveform=Mu("/servers/user/MU/simulation/ampTest/spectre/schematic/psf")
```

```
=> srrWave:0x37627330
```

The following example plots the waveform object `MuWaveform`.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(MuWaveform)
                ?expr list("Mu")
                ?color list("y12")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("o")
                )
=> t
```



mu_prime

```
mu_prime(
     o_s11Waveform
     o_s12Waveform
     o_s21Waveform
     o_s22Waveform
     )
     => 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_s11Waveform	Waveform object representing the S-Parameter `s11`.
o_s12Waveform	Waveform object representing the S-Parameter `s12`.
o_s21Waveform	Waveform object representing the S-Parameter `s21`.
o_s22Waveform	Waveform object representing the S-Parameter `s22`.

Value Returned

o_waveform	Waveform object representing the minimum distance between the center of the unit Smith chart and the source unstable region.
`nil`	Waveform object cannot be created because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results of S-Parameters analysis stored in the specified results directory.

```
openResults("/servers/user/MU/simulation/ampTest/spectre/schematic/psf")
=> "/servers/user/MU/simulation/ampTest/spectre/schematic/psf"
```

The following example returns the waveform object representing S-Parameter `s11`.

```
s11=aaSP(1 1)
=> srrWave:0x37627020
```

The following example returns the waveform object representing S-Parameter `s12`.

```
s12=aaSP(1 2)
=> srrWave:0x37627030
```

The following example returns the waveform object representing S-Parameter `s21`.

```
s21=aaSP(2 1)
=> srrWave:0x37627040
```

The following example returns the waveform object representing S-Parameter `s22`.

```
s22=aaSP(2 2)
=> srrWave:0x37627050
```

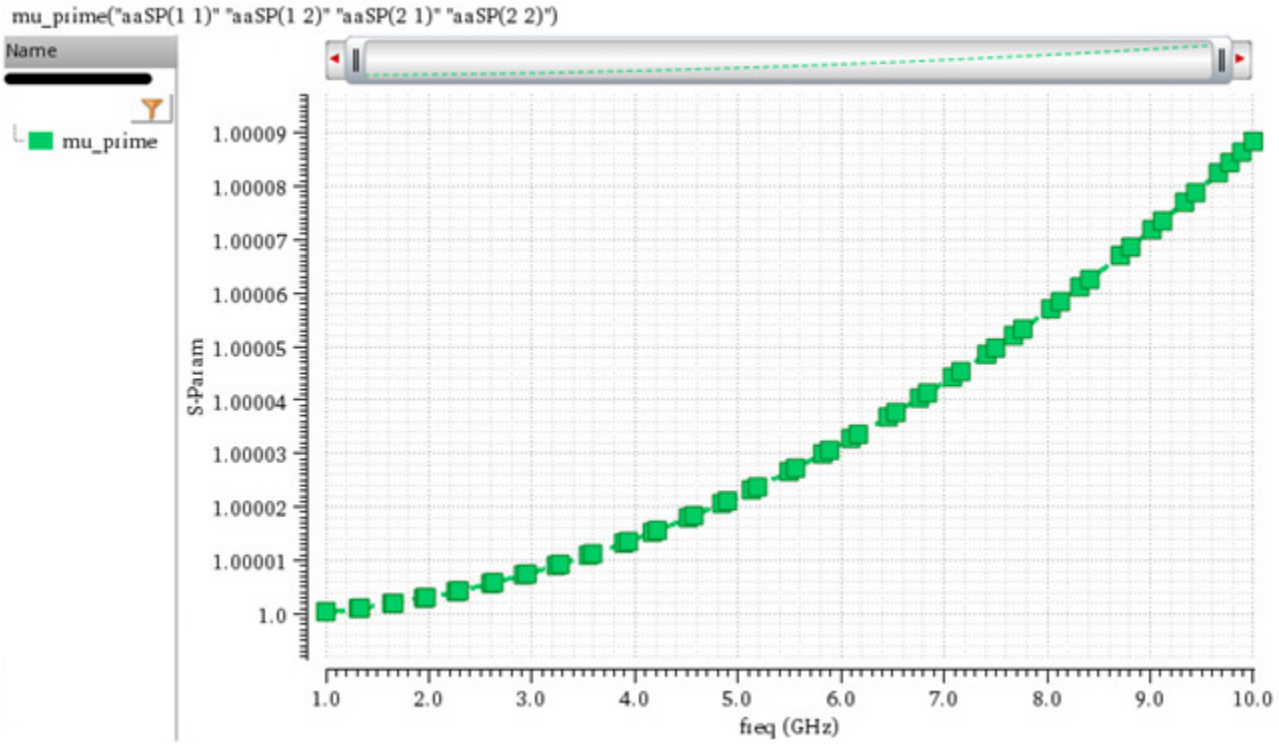
The following example creates a waveform object `mu_primeWaveform`, representing the minimum distance between the center of the unit Smith chart and the source unstable region.

```
mu_primeWaveform=mu_prime(s11 s12 s21 s22)
srrWave:0x37627210
```

The following example plots the waveform object `mu_primeWaveform`.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(mu_primeWaveform)
                ?expr list("mu_prime")
                ?color list("y18")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list(3)
                )
```

```
=> t
```



Mu_prime

```
Mu_prime(
     t_resultsDirectory
     )
     => 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

t_resultsDirectory

Path to the results directory that contains results of S-Parameter analysis.

Value Returned

o_waveform	Waveform object represnting the minimum distance between the origin of the unit Smith chart and the load unstable region.
nil	Waveform object cannot be created because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results of S-Parameters analysis stored in the specified results directory.

```
openResults("/servers/user/MU/simulation/ampTest/spectre/schematic/psf")
=> "/servers/user/MU/simulation/ampTest/spectre/schematic/psf"
```

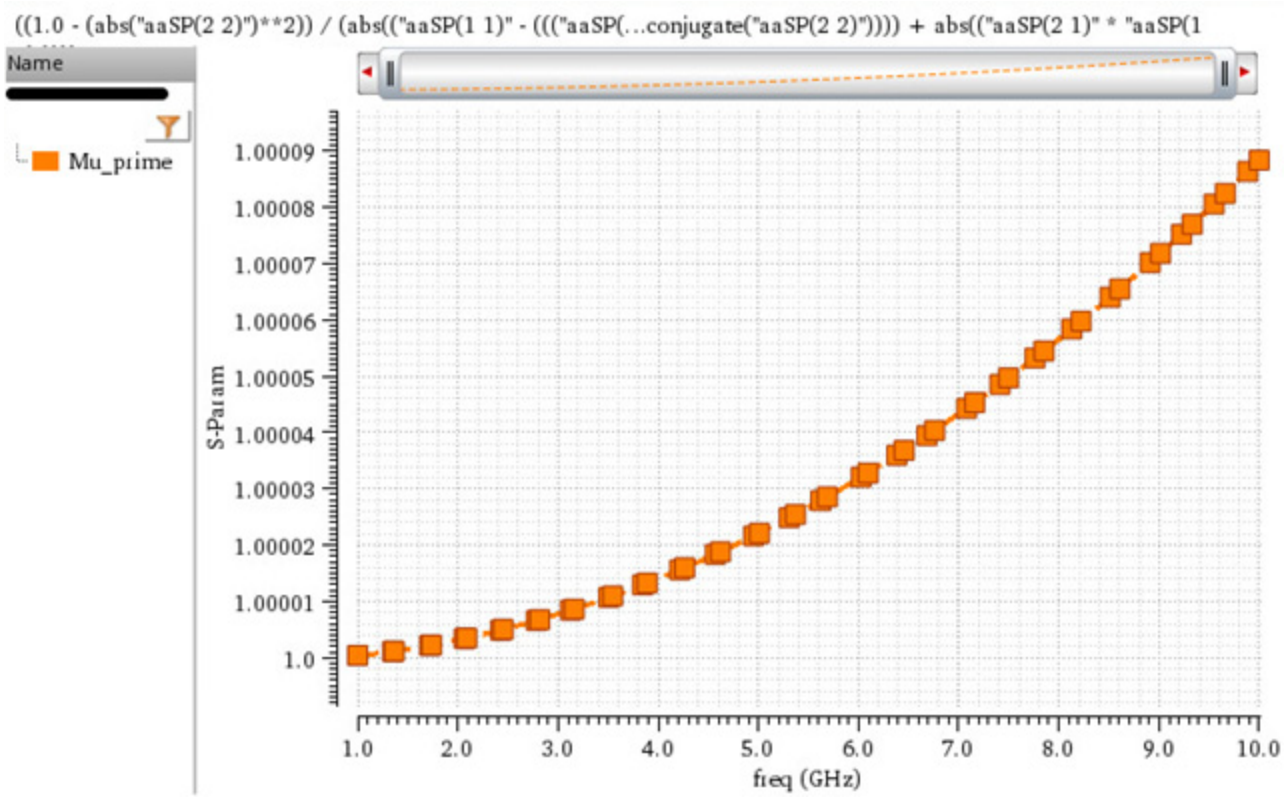
The following example creates a waveform object `Mu_primeWaveform`, representing the minimum distance between the center of the unit Smith chart and the source unstable region.

```
Mu_primeWaveform=Mu_prime("/servers/userMU/simulation/ampTest/spectre/schematic/
psf")
```

```
=> srrWave:0x37627450
```

The following example plots the waveform object `Mu_primeWaveform`.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(Mu_primeWaveform)
                ?expr list("Mu_prime")
                ?color list("y6")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list(3)
                )
=> t
```



normalQQPValue

```
normalQQPValue(
     o_waveform
     )
     => n_pValue / nil
```

Description

Returns the p-value of a quantile-quantile (QQ) plot.

A QQ plot draws the correlation between a given sample and the normal distribution.

A small p-value indicates that the distribution is unlikely to be normal, while a large p-value indicates that the distribution is close to normal.

Arguments

o_waveform	Waveform representing the QQ plot whose p-value is to be calculated.

Value Returned

n_pValue	The p-value of the QQ plot.
nil	The p-value cannot be calculated because of an error.

Examples

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/distribution.psf")
=> "/servers/user/distribution.psf"
```

The following example lists the results contained in the results directory.

```
results()
=> ("tran")
```

The following example selects the `tran` results of the results directory.

```
selectResults('tran)
=> stdobj@0x2a9de759
```

The following example creates a waveform object for the signal `v3net` stored in the `tran` results.

```
waveform=v("v3net")
=> srrWave:0x3232c030
```

The following example calculates the p-value of the specified waveform.

```
normalQQPValue(waveform)
=> 0.7149
```

numConv

```
numConv(
     t_inputNumber
     t_format
     g_needPrefix
     )
     => t_outputNumber / nil
```

Description

Converts an input number to the specified format.

Arguments

t_inputNumber	The number to be converted into the specified format. You must specify the number in a string format.
t_format	Format in which you want to convert the specified number. Valid values are: ■ `bin`: Converts the number into binary format. ■ `dec`: Converts the number into decimal format. ■ `hex`: Converts the number into hexadecimal format. ■ `oct`: Converts the number into octal format. ■ `sdec`: Converts the number into signed decimal format. ■ `udec`: Converts the number into unsigned decimal format.
g_needPrefix	Specifies whether to add a prefix to the output number to indicate its format. Valid values are: ■ `t`: A prefix is added to the output. The prefix depends on the specified format in which you want to convert the input number. For example, prefix `0` is added to the output if the specified format is octal. Prefix `0b` indicates that the output format is binary. Prefix `0x` indicates that the output format is hexadecimal. ■ `nil`: Prefix is not added to the output.

Value Returned

t_outputNumber	Output number converted in the specified format. The output value is returned in the string format.
nil	Number cannot be converted because of an error.

Examples

The following example converts an input number 100 into hexadecimal format. Note that a prefix 0x is added to the output because the argument *g_needPrefix* is set to t.

```
numConv("100" "hex" t)
=> "0x64"
```

The following example converts an input number 100 into hexadecimal format. Note that no prefix is added to the output because the argument *g_needPrefix* is set to nil.

```
numConv("100" "hex" nil)
=> "64"
```

The following example converts an input number 100 into binary format. Note that a prefix 0b is added to the output because the argument *g_needPrefix* is set to t.

```
numConv("100" "bin" t)
=> "0b1100100"
```

The following example converts an input number 100 into binary format. Note that no prefix is added to the output because the argument *g_needPrefix* is set to nil.

```
numConv("100" "bin" nil)
=> "1100100"
```

OS

```
OS(
     t_instanceName
     t_parameterName
     )
     => o_waveform / nil
```

Description

Returns the specified device parameter of the an instance from DC analysis data.

Important

To plot or evaluate an OS expression, you must add an OS expression or an oppoint expression to the outputs in ADE Assembler or ADE Explorer and run a simulation. This is because the operating points are not saved when you run a DC sweep or transient simulation, unless they are explicitly used in an expression or an oppoint output.

Name	Type	Details
Filter	Filter	Filter
OTi	expr	OT("/V1" "i")
OSpwr	expr	OS("/V1" "pwr")
V1	oppoint	/V1 i v pwr

Arguments

t_instanceName	Name of the instance for which the device parameter is to be calculated.
t_parameterName	Name of the device parameter to be calculated.

Value Returned

o_waveform	Waveform representing the calculated device parameter of the specified instance.
nil	The device parameter cannot be calculated because of an error.

Examples

The following example opens simulation results of transient and DC analysis stored in the specified results directory.

```
openResults("/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/test/psf")
=> "/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/test/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
= > tran(tranOp dcOp dcOpInfo dc
    model instance output
    designParamVals primitives subckts
    variables
)
```

The following example selects the results of DC analysis.

```
selectResults('dc)
=> stdobj@0x316292a8
```

The following example lists the output signals saved in the selected result.

```
outputs()
=> ("/D0:v" "/V1:v" "/net3" "/emitter" "/net021"
    "/net17" "/varactor1" "/net30" "/collector" "/base"
    "/net8" "/out" "/IPRB0/PLUS" "/L0/PLUS" "/L1/PLUS"
    "/V0/PLUS" "/V1/PLUS" "/V2/PLUS" "V1:pwr" "/C0:cap"
    "/D0:i" "/IPRB0:i" "/V1:i" "/D0:gd"
)
```

The following example creates a waveform object `wave` that represents the waveform of device parameter `pwr` of the instance `V1` for DC analysis.

```
wave=OS("/V1" "pwr")
=> srrWave:0x35817010
```

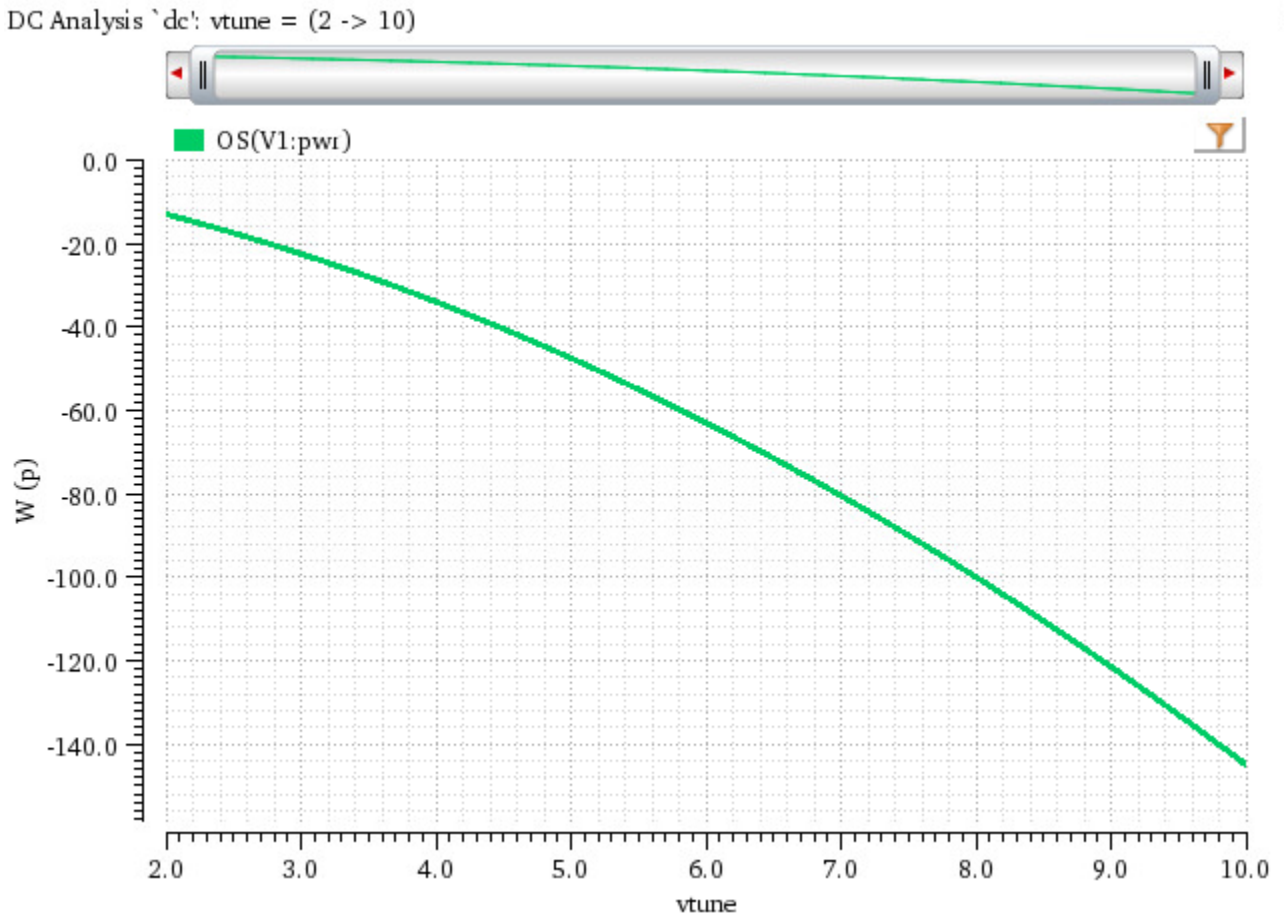
The following example creates a Waveform window `win` and returns its window ID.

```
win=awvCreatePlotWindow()
=> window:3
```

The following example plots the waveform object `wave` in the Waveform window `win`.

```
awvPlotWaveform(
                win
                list(wave)
```

```
                ?expr list("OS(V1:pwr)")
                ?color list("y18")
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                )
=> t
```



OT

```
OT(
     t_instanceName
     t_parameterName
     )
     => o_waveform / nil
```

Description

Returns the specified device parameter of the given instance from the transient analysis data.

Important

To plot or evaluate an OT expression, you must add an OT expression or an oppoint expression to the outputs in ADE Assembler or ADE Explorer and run a simulation. This is because the operating points are not saved when you run a DC sweep or transient simulation, unless they are explicitly used in an expression or an oppoint output.

Name	Type	Details
Filter	Filter	Filter
OTi	expr	OT("/V1" "i")
OSpwr	expr	OS("/V1" "pwr")
V1	oppoint	/V1 i v pwr

Arguments

t_instanceName	Name of the instance for which the device parameter is to be calculated.
t_parameterName	Name of the device parameter to be calculated.

Value Returned

o_waveform	Waveform representing the calculated device parameter of the specified instance.
nil	The device parameter cannot be calculated because of an error.

Examples

The following example opens simulation results of transient and DC analysis stored in the specified results directory.

```
openResults("/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/test/psf")
=> "/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/test/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
= >
tran(tranOp dcOp dcOpInfo dc
    model instance output
    designParamVals primitives subckts
    variables
)
```

The following example selects the results of transient analysis.

```
selectResults('tran)
=> stdobj@0x31de7248
```

The following example lists the output signals saved in the selected result.

```
outputs()
=>
("/D0:v" "/V1:v" "/net3" "/emitter" "/net021"
    "/net17" "/varactor1" "/net30" "/collector" "/base"
    "/net8" "/out" "/IPRB0/PLUS" "/L0/PLUS" "/L1/PLUS"
    "/V0/PLUS" "/V1/PLUS" "/V2/PLUS" "V1:pwr" "/C0:cap"
    "/D0:i" "/IPRB0:i" "/V1:i" "/D0:gd"
)
```

The following example creates a waveform object `wave` that represents the waveform of device parameter `i` of the instance `V1` for the transient analysis.

```
wave=OT("/V1" "i")
=> srrWave:0x35817020
```

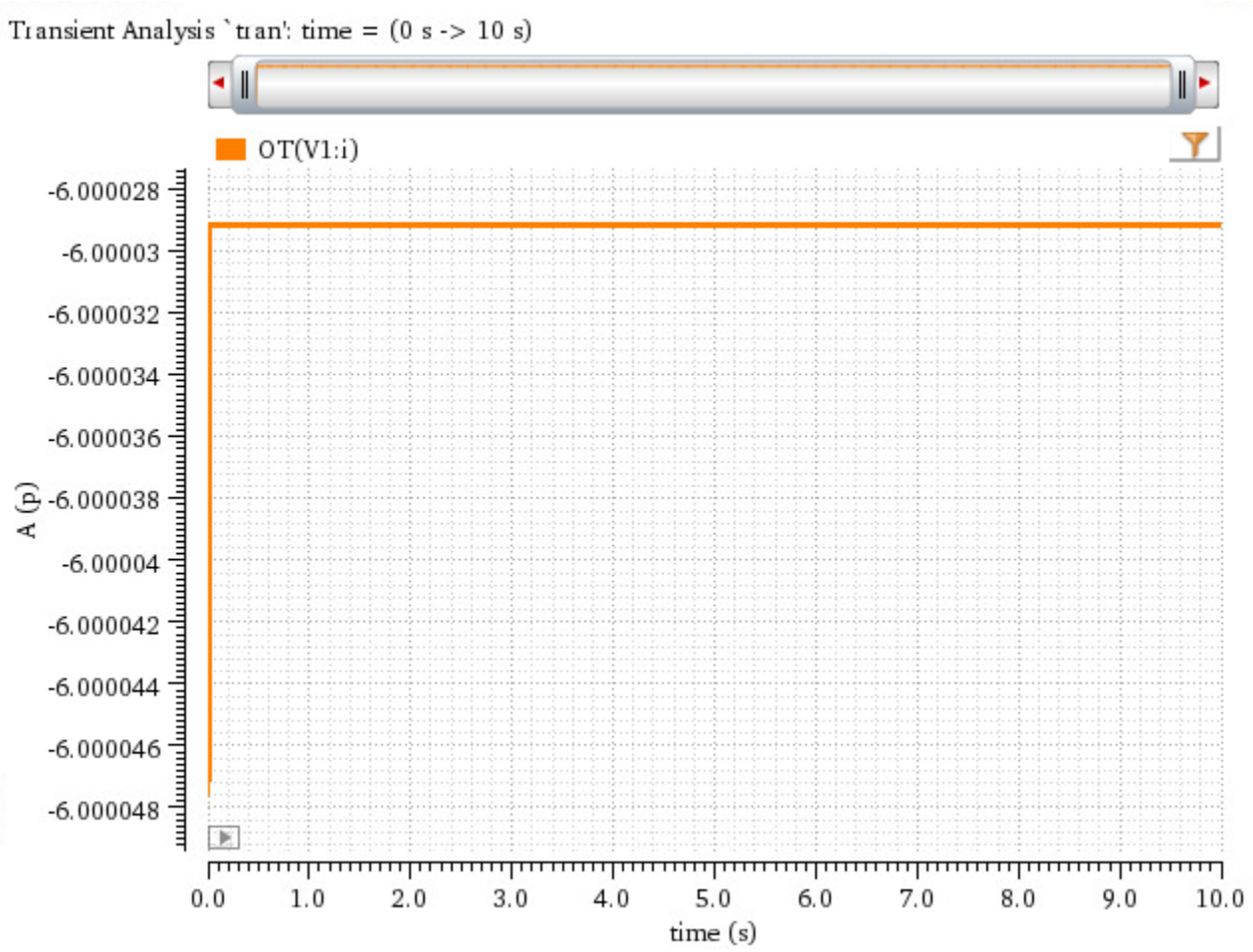
The following example creates a waveform window and returns its window ID.

```
win=awvCreatePlotWindow()
=> window:4
```

The following example plots the waveform object `wave` in the Waveform window `win`.

```
awvPlotWaveform(
```

```
                win
                list(wave)
                ?expr list("OT(V1:i)")
                ?color list("y6")
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                )
=> t
```



pvifreq

```
pvifreq(
     s_analysis
     t_pos
     t_neg
     t_branch1
     t_branch2
     [ 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_analysis`	Name of the analysis. Default value is `hb`.
`t_pos`	Positive node or net from the schematic or from the Results Browser. You can also specify a voltage value for this argument.
`t_neg`	Negative node or net from the schematic or from the Results Browser. You can also specify a voltage value for this argument.
`t_branch1`	First branch name on the schematic or the signal name from the Results Browser.
`t_branch2`	Second branch name on the schematic or the signal name from the Results Browser.
`x_freq`	Frequency for which you want to plot the results. It is an optional argument. You can specify any integer value. Default value is `nil`.

Value 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	Spectral power cannot be calculated because of an error.

Examples

The following example opens simulation results of hb-hbnoise analysis stored in the specified results directory.

```
openResults("/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/test/psf")
=> "/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/test/psf"
```

The following example lists the available results in the currently open results directory.

```
results()
=>
(xf hb_fi hb_fd hb_td hbnoise_sample_hm0
    hbnoise_sample_hm1 model instance output designParamVals
    primitives subckts variables
)
```

The following example selects the result hb_fi.

```
selectResults('hb_fi)
=> stdobj@0x31533260
```

The following example lists the output signals stored in the selected result.

```
outputs()
=>
("/net3" "/emitter" "/net021" "/net17" "/varactor1"
    "/net30" "/collector" "/base" "/net8" "/out"
    "/IPRB0/PLUS" "/L0/PLUS" "/L1/PLUS" "/V0/PLUS" "/V1/PLUS"
    "/V2/PLUS"
)
```

The following example creates a waveform object wave that represents the spectral power calculated for the arguments specified for the function pvifreq.

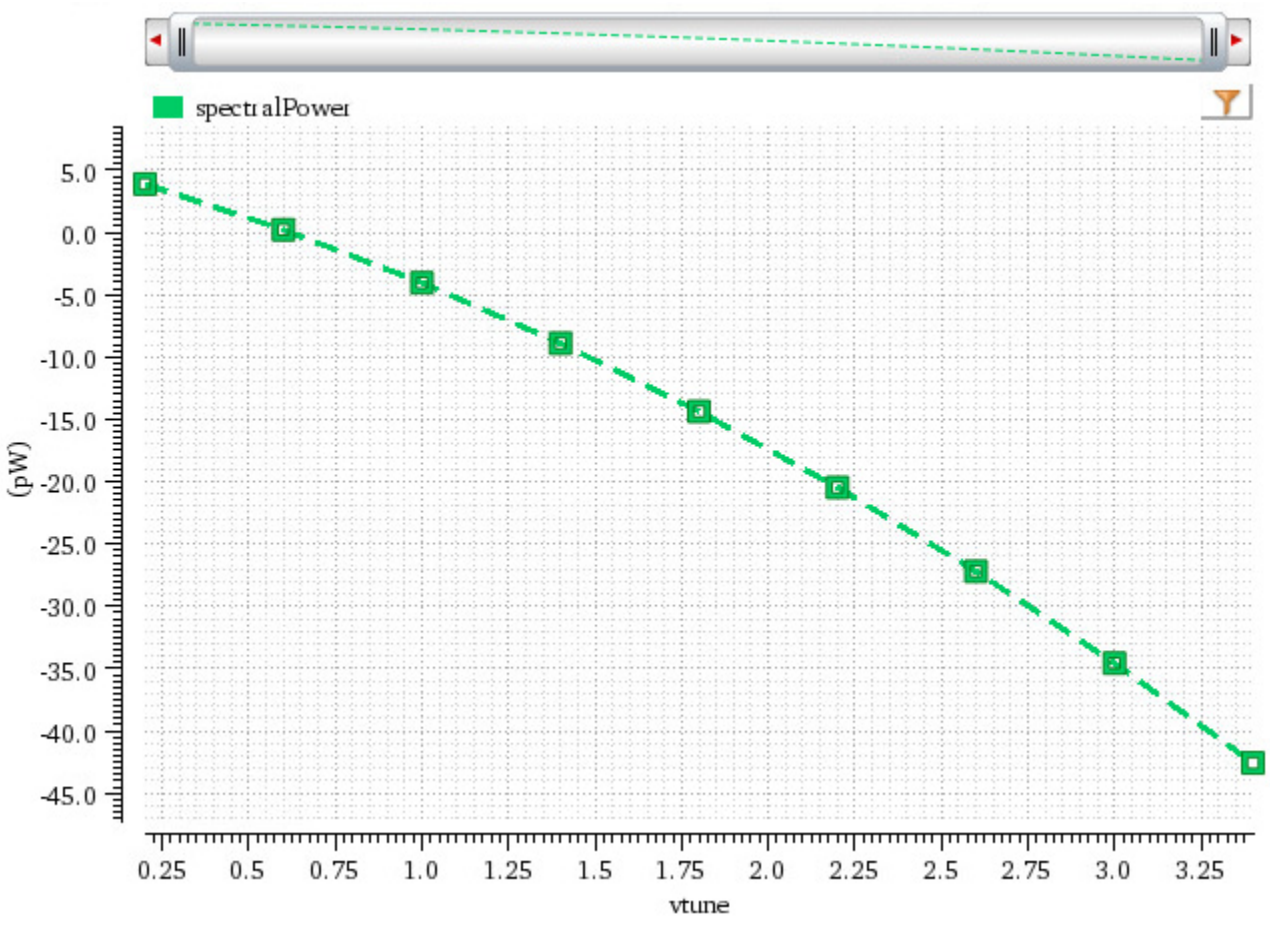
```
wave=pvifreq("hb" "/net30" "/net021" "/V1/PLUS" "/V1/MINUS" 50)
=> srrWave:0x35728bb0
```

The following example creates a waveform window and returns its window ID.

```
win=awvCreatePlotWindow()
=> window:4
```

The following example plots the waveform represented by the waveform object `wave`.

```
awvPlotWaveform(
                win
                list(wave)
                ?expr list("spectralPower")
                ?color list("y18")
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list(5)
                )
=> t
```



pvrfreq

```
pvrfreq(
     s_analysis
     t_pos
     t_neg
     x_resistance
     [ 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_analysis`	Name of the analysis. Default value is `hb`.
`t_pos`	Positive node or net from the schematic or from the Results Browser. You can also specify a voltage value for this argument.
`t_neg`	Negative node or net from the schematic or from the Results Browser. You can also specify a voltage value for this argument.
`x_resistance`	Value of the resistance. You can specify any integer or a floating-point number.
`x_freq`	Frequency for which you want to plot the results. It is an optional argument. You can specify any integer value. Default value is `nil`.

Value 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	Spectral power cannot be calculated because of an error.

Examples

The following example opens simulation results of hb-hbnoise analysis stored in the specified results directory.

```
openResults("/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/test/psf")
=> "/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/test/psf"
```

The following example lists the available results in the currently open results directory.

```
results()
=>
(xf hb_fi hb_fd hb_td hbnoise_sample_hm0
    hbnoise_sample_hm1 model instance output designParamVals
    primitives subckts variables
)
```

The following example selects the result hb_fi.

```
selectResults('hb_fi)
=> stdobj@0x31533260
```

The following example lists the output signals stored in the selected result.

```
outputs()
=>
("/net3" "/emitter" "/net021" "/net17" "/varactor1"
    "/net30" "/collector" "/base" "/net8" "/out"
    "/IPRB0/PLUS" "/L0/PLUS" "/L1/PLUS" "/V0/PLUS" "/V1/PLUS"
    "/V2/PLUS"
)
```

The following example creates a waveform object wave that represents the spectral power calculated for the arguments specified for the function pvrfreq.

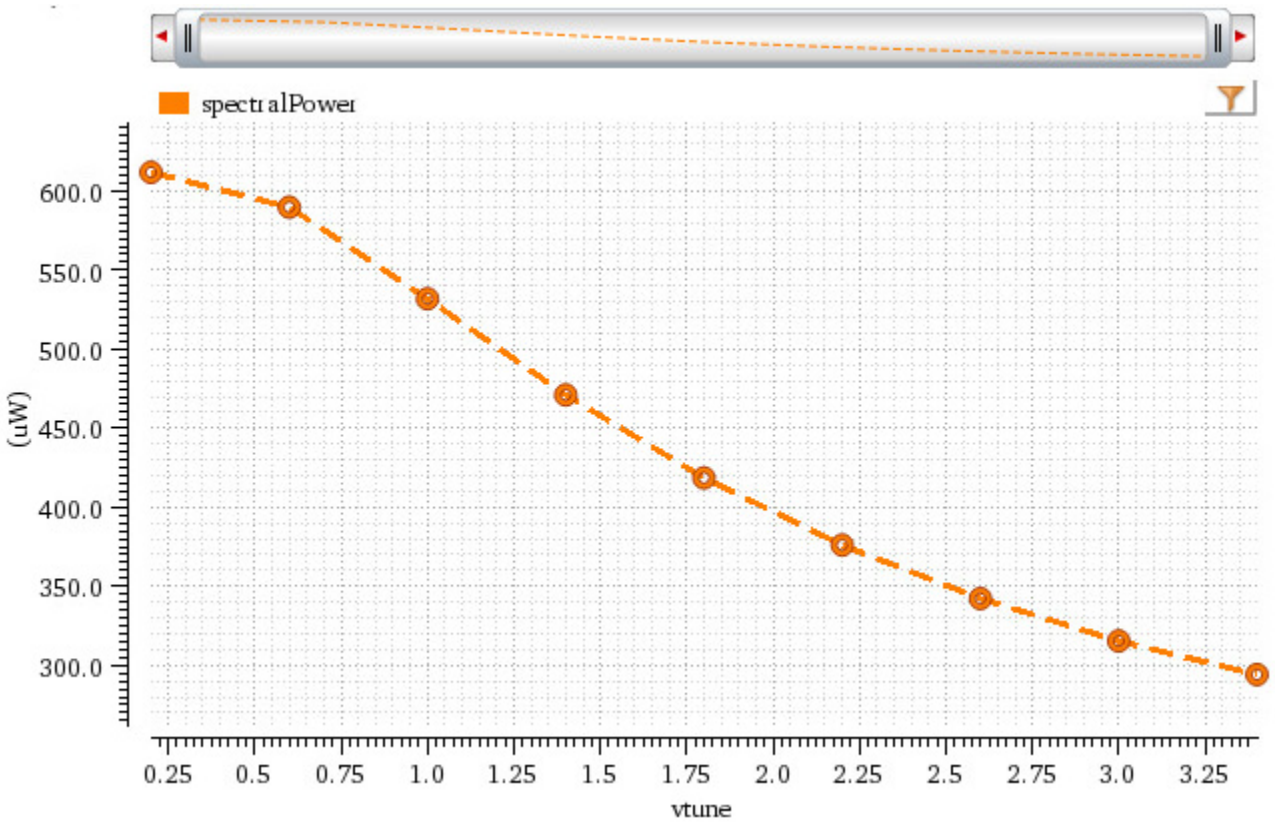
```
wave=pvrfreq("hb" "/net30" "/net021" 20 4e9)
=> srrWave:0x35729b00
```

The following example creates a waveform window and returns its window ID.

```
win=awvCreatePlotWindow()
=> window:3
```

The following example plots the waveform represented by the waveform object `wave`.

```
awvPlotWaveform(
                win
                list(wave)
                ?expr list("spectralPower")
                ?color list("y6")
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("o")
                )
=> t
```



skewness

```
skewness(
     o_waveform
     )
     => n_skewness / nil
```

Description

Calculates the skewness of the specified waveform.

Skewness is a measurement of symmetry or, more precisely, the lack of symmetry. A perfectly symmetric waveform has a skewness value of zero.

Arguments

o_waveform	Waveform whose skewness is to be calculated.

Value Returned

n_skewness	Skewness value of the specified waveform.
`nil`	Skewness value cannot be calculated because of an error.

Examples

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/distribution.psf")
=> "/servers/user/distribution.psf"
```

The following example lists the results contained in the results directory.

```
results()
=> ("tran")
```

The following example selects the `tran` results of the results directory.

```
selectResults('tran)
=> stdobj@0x2a9de759
```

The following example creates a waveform object for the signal `v1net` stored in the `tran` results.

```
waveform=v("v1net" ?result "tran")
```

```
=> srrWave:0x3232c020
```

The following example calculates the skewness value of the specified waveform.

```
skewness(waveform)
=> 0.1106943
```

swapSweep

```
swapSweep(
     o_waveform
     t_sweepVariable
     n_xValue
     )
     => o_waveform / nil
```

Description

Swaps x-axis values with the specified sweep variable.

Arguments

o_waveform	Waveform whose x-axis values you want to swap with a sweep variable.
t_sweepVariable	Name of the sweep variable.
n_xValue	X-axis value at which the specified waveform is swapped with the sweep variable.

Value Returned

o_waveform	Waveform that has x-axis values swapped with the sweep variable.
nil	X-axis values of the specified waveform cannot be swapped because of an error.

Examples

The following example creates a Waveform window and returns its ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/sweeptran.raw")
=> "/servers/user/design/sweeptran.raw"
```

The following example selects the `tran` result from the specified results directory.

```
selectResult('tran)
```

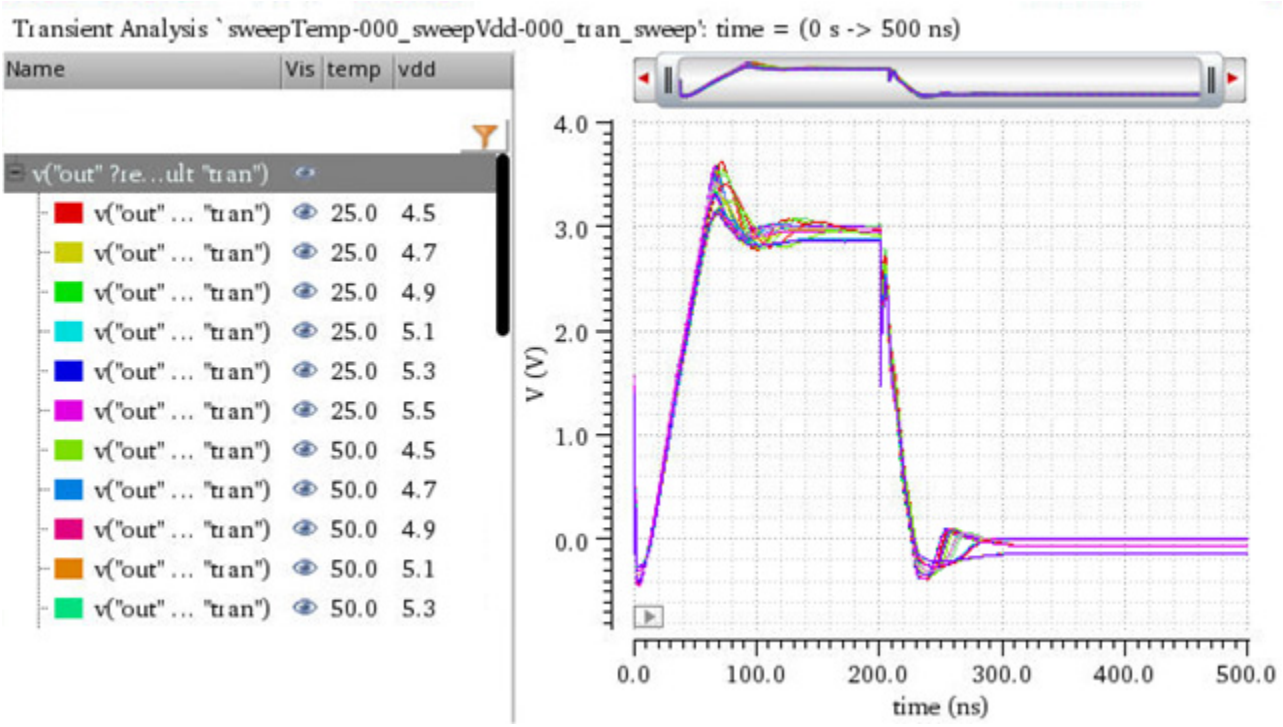
```
=> stdobj@0x3295cbc0
```

The result `tran` contains a waveform out generated by simulating parametric data using the sweep variables, `vdd` and `temp`. The following example stores the waveform object ID of the signal `out` to a variable `waveform`.

```
waveform=v("out")
=> srrWave:0x36c69020
```

The following example plots the waveform `out` in the Waveform window that you created using the function `awvCreatePlotWindow`.

```
plot waveform
=> t
```



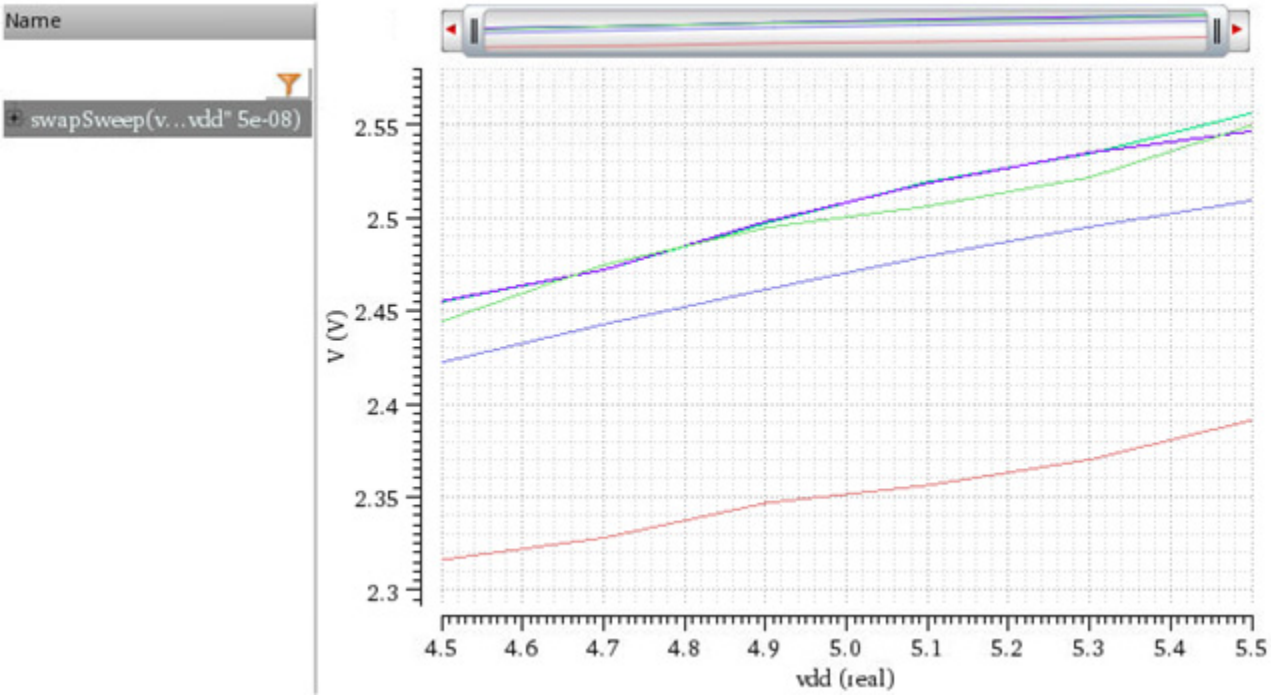
The following example swaps x-axis values of waveform `out` at x=`50ns` with the sweep variable `vdd`.

```
swapWaveformVdd=swapSweep(waveform "vdd" 50ns)
=> srrWave:0x36c690d0
```

The following example plots the swapped waveform.

```
plot swapWaveformVdd
=> t
```

Note that the output waveform has x axis swapped with the specified sweep variable `vdd`.



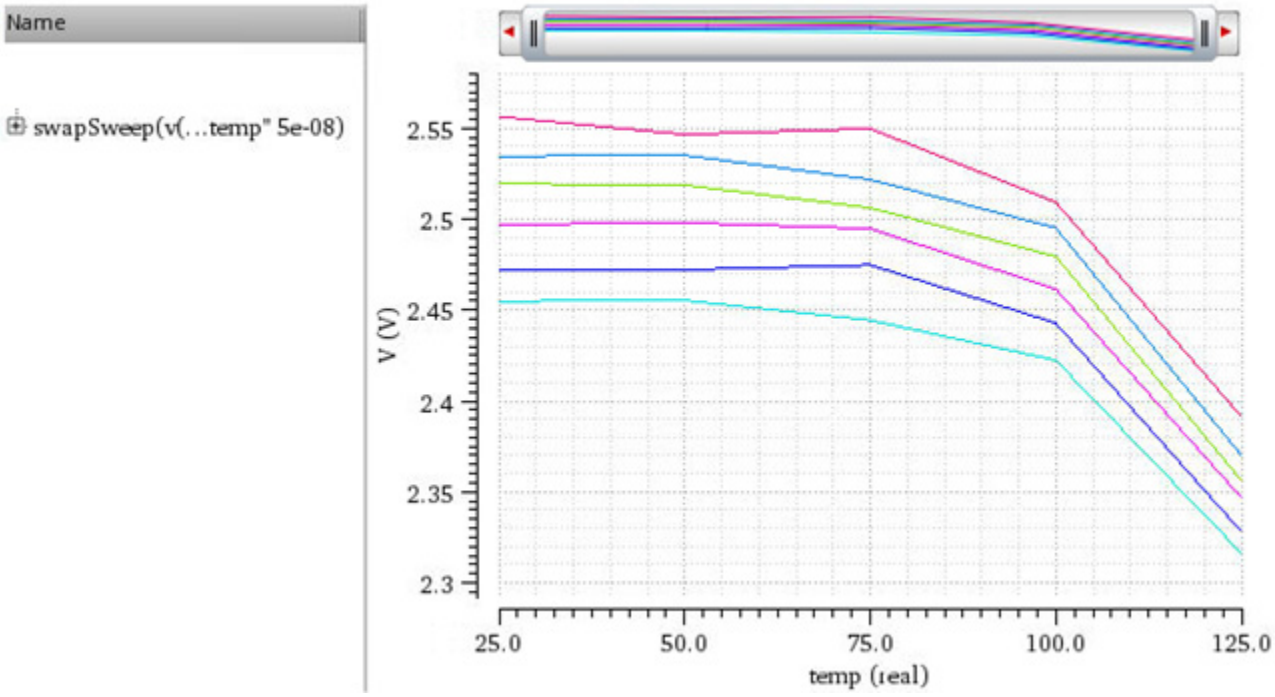
The following example swaps x-axis values of waveform `out` at x=`50ns` with the sweep variable `temp`.

```
swapWaveformTemp=swapSweep(waveform "temp" 50ns)
=> srrWave:0x36c69270
```

The following example plots the swapped waveform.

```
plot swapWaveformTemp
=> t
```

Note that the output waveform has x axis swapped with the specified sweep variable `temp`.



topBaseLine

```
topBaseLine(
     o_waveform
     )
     => l_values / nil
```

Description

Returns the topline and baseline values of the specified transient waveform.

Arguments

o_waveform	Waveform whose topline and baseline values are to be calculated.

Value Returned

l_values	Topline and baseline values of the specified transient waveform.
nil	Topline and baseline values cannot be calculated because of an error.

Examples

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

The `tran-tran` result of the results directory `ampsim.raw` contain a transient signal `out`.

```
waveform=v("out" ?result "tran-tran")
=> srrWave:0x34930020
```

The following example calculates the topline and baseline value of the transient signal `out`.

```
topBaseLine(waveform)
=> (2.926673 -0.03882059)
```

The first value in the list is the topline value and the second value is the baseline value.

topLine

```
topLine(
     o_waveform
     )
     => l_values / nil
```

Description

Returns the topline value of the specified transient waveform.

Arguments

o_waveform	Waveform whose topline value is to be calculated.

Value Returned

l_values	Topline value of the specified transient waveform.
nil	Topline value cannot be calculated because of an error.

Examples

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

The `tran-tran` result of the results directory `ampsim.raw` contain a transient signal `out`.

```
waveform=v("out" ?result "tran-tran")
=> srrWave:0x34930020
```

The following example calculates the topline value of the transient signal `out`.

```
topLine(waveform)
=> 2.926673
```

triggeredDelay

```
triggeredDelay(
     o_signal1
     o_signal2
     n_threshold1
     s_edgeType1
     n_threshold2
     s_edgeType2
     [ ?multiple g_multiple]
     [ ?nth x_nth ]
     [ ?periodicity x_periodicity ]
     [ ?tol1 n_tolerance1 ]
     [ ?tol2 n_tolerance2 ]
     [ ?xName s_xName ]
     )
     => o_waveform / n_value / nil
```

Description

Calculates the delay from the trigger point on the edge (either rising or falling) of a triggering signal to the next edge (either rising or falling) of the target signal.

Arguments

o_signal1	Triggering signal.
o_signal2	Target signal.
n_threshold1	Threshold value of the triggering signal.
s_edgeType1	Direction of the crossing event for the triggering signal. ■ `rising`: Directs the function to look for crossings events where the y values are increasing. ■ `falling`: Directs the function to look for crossings events where the y values are decreasing. ■ `either`: Directs the function to look for crossings events in either direction.
n_threshold2	Threshold value of the target signal.

s_edgeType2 Direction of the crossing event for the target signal.

- `rising`: Directs the function to look for crossings events where the y values are increasing.
- `falling`: Directs the function to look for crossings events where the y values are decreasing.
- `either`: Directs the function to look for crossings events in either direction.

`?multiple` *g_multiple*

Specifies whether to retrieve only one occurrence of a delay event for the waveform (single), or all occurrences of an overshoot for the waveform.

Valid values are:

- `t`: Returns the waveform of measured delay starting from the n^{th} edge.
- `nil`: Returns a single delay at the n^{th} edge.

Default value is `t`.

`?nth` *x_nth* Edge number of the triggering signal from which delay is to be calculated.

Default value is `1`, which indicates that delay is calculated from the first edge of the triggering signal.

`?periodicity` *x_periodicity*

Periodic interval for the triggering signal.

Default value is `1`.

`?tol1` *n_tolerance1*

Tolerance value to detect the threshold crossings for the triggering signal.

Default value is `0.0`.

`?tol2` *n_tolerance2*

Tolerance value to detect the threshold crossings for the target signal.

Default value is `0.0`.

?xName *s_xName*	Specifies whether to retrieve delay data against trigger time, target time (or another x-axis parameter for non-transient data), or cycle. Possible values are `trigger`, `target`, and `cycle`. Default value is `trigger`.

Value Returned

o_waveform	Waveform if the `?multiple` argument is set to `t`.
n_value	
`nil`	because of an error.

Examples

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/triggeredDelay.psf")
=> "/servers/user/triggeredDelay.psf"
```

The following example lists the results stored in the directory.

```
results()
=> ("tran")
```

The following example selects the `tran` results stored in the specified results directory triggeredDelay.psf.

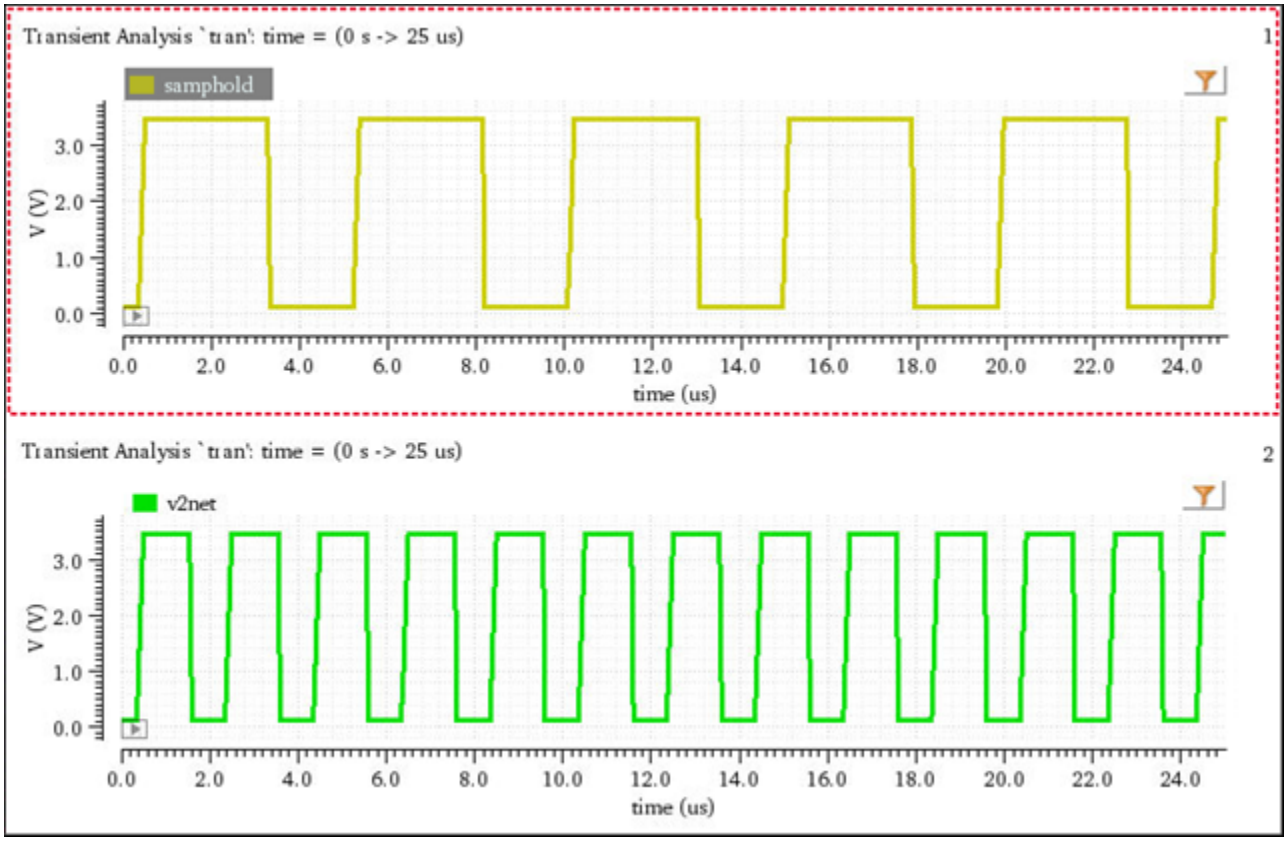
```
selectResults('tran)
=> stdobj@0x2d4000c8
```

The `tran` results of the results directory `triggeredDelay.psf` contain two signals, `samphold` and `v2net`.

The following examples plot signals `samphold` and `v2net` in separate subwindows.

```
awvPlotSignals('(("/servers/user/triggeredDelay.psf" (("tran" ("samphold")))))
?plotStyle "New Subwindow")
=> t
awvPlotSignals('(("/servers/user/triggeredDelay.psf" (("tran" ("v2net")))))
?plotStyle "New Subwindow")
```

```
=> t
```



The following example calculates delay from the trigger point on the first rising edge of the triggering signal, `samphold`, to the next edge (either rising or falling) of the target signal, `v2net`.

```
delay=triggeredDelay(v("samphold" ?result "tran") v("v2net" ?result "tran") 1.7
"rising" 1.7 "either" ?multiple t ?xName "target")
=> srrWave:0x2fbe4070
```

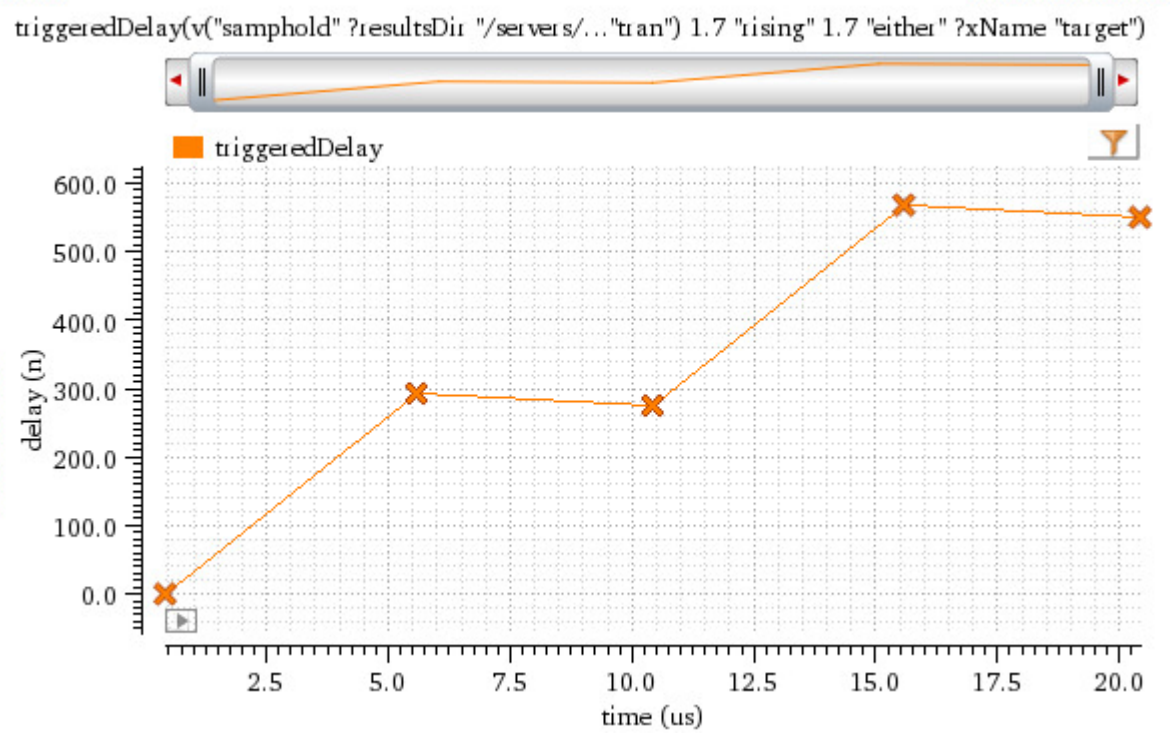
The following example creates a new Waveform window.

```
awvCreatePlotWindow()
=> window:5
```

The following example plots the waveform represented by delay in the newly created Waveform window.

```
awvAppendWaveform(awvGetCurrentWindow() list(delay) ?expr list("triggeredDelay")
?color list("y6") ?index list(2) ?lineType list("line") ?showSymbols list(t)
?dataSymbol list("x"))
```

```
=> t
```



valueAt

```
valueAt(
     o_waveform
     n_xValue
     [ ?extrapolate g_extrapolate ]
     )
     => n_yValue / nil
```

Description

Returns the y-axis value of a waveform at the specified x-axis value.

Arguments

o_waveform	Waveform ID.
n_xValue	X-axis value at which the y-axis value is to be returned for the specified waveform.
?extrapolate *g_extrapolate*	Specifies whether to use extrapolation to calculate the estimated y-axis value if the specified x-axis value does not exist. Valid values are: ■ `t`: Extrapolation is used. ■ `nil`: Extrapolation is not used.

Value Returned

n_yValue	Y-axis value of the waveform at the specified x-axis value.
`nil`	Y-axis value cannot be returned because of an error.

Examples

The following example returns the waveform ID of the eye diagram `eye` created from the `jitter` signal.

```
eye=eyeDiagram(v("jitter" ?result "tran-tran" ?resultsDir "/servers/user/design/
prbs.raw") 200n 400u 2*40n   ?autoCenter t )
=> srrWave:0x336321f0
```

The following example returns the waveform ID of the curve representing the left-side bit error rate (BER) curve for the eye diagram `eye`.

```
waveform=eyeBERLeft(eye 200n 400u 80n 0.5 100)
=> srrWave:0x336322e0
```

The following example returns the last value at which the waveform returned by the function `eyeBERLeft` ends on x axis.

```
lastVal(waveform)
=> 2.353111e-08
```

Indicates that the `waveform` ends at `x=23.53111ns`.

The following example returns the y-axis value of `waveform` at `x=23.5ns`.

```
valueAt(waveform 23.5ns)
=> 5.096838e-07
```

The following example returns `nil` because the specified x-axis value `24.5ns` is greater than `23.53111ns`, which is the last value of `waveform` on x axis.

```
valueAt(waveform 24.5ns)
*Warning* : Index value out of range for waveform(nil)
=> nil
```

To return the y-axis value of waveform at x=24.5ns, set the ?`extrapolate` argument to `t`.

```
valueAt(waveform 24.5ns ?extrapolate t)
=> 9.066262e-08
```

vvDisplayCalculator

```
vvDisplayCalculator(
     [ t_expression ]
     )
     => t / nil
```

Description

Opens Calculator within a window. The function also displays the expression, if specified, in the buffer.

Arguments

t_expression	Expression to be displayed in the buffer. This is an optional argument.

Value Returned

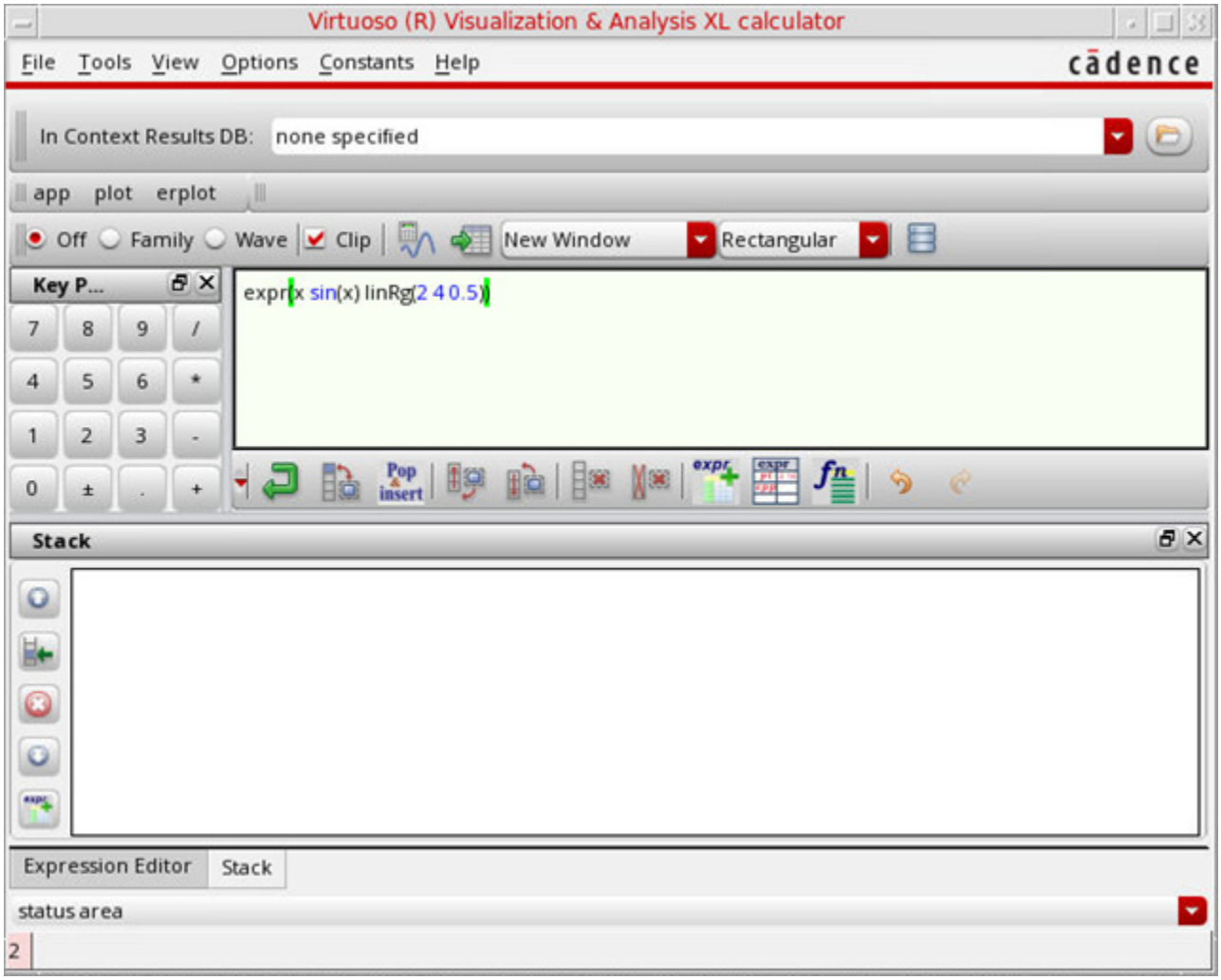
t	Calculator opens successfully.
nil	Calculator cannot be opened because of an error.

Examples

The following example opens Calculator and displays the specified expression in the Buffer.

```
vvDisplayCalculator("expr(x sin(x) linRg(2 4 0.5))")
=> calSetBuffer("expr(x sin(x) linRg(2 4 0.5))")
=> t
```

```
=> t
```



The following example opens Calculator.

```
vvDisplayCalculator()
=> t
```

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 that uses y axes of the specified input waveforms as its x and y axes. 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_waveX`	Input waveform whose y axis you want to use as the x axis of the output waveform.
`?y o_waveY`	Input waveform whose y axis you want to use as the y axis of the output waveform.
`?xName t_xName`	Name of the x axis.
`?xUnits g_xUnits`	Specifies whether to show units on x axis.
`?yName t_yName`	Name of the y axis.
`?yUnits g_yUnits`	Specifies whether to show units on y axis.

Value Returned

`o_waveform`	Waveform object of the output waveform created using the specified input waveforms.
`nil`	Output waveform cannot be created because of an error.

Examples

The following example creates a Waveform window and returns its window ID.

```
awvCreatePlotWindow()
=> window:3
```

The following example opens simulation results stored in the specified directory.

```
openResults("/servers/user/design/ampsim.raw")
=> "/servers/user/design/ampsim.raw"
```

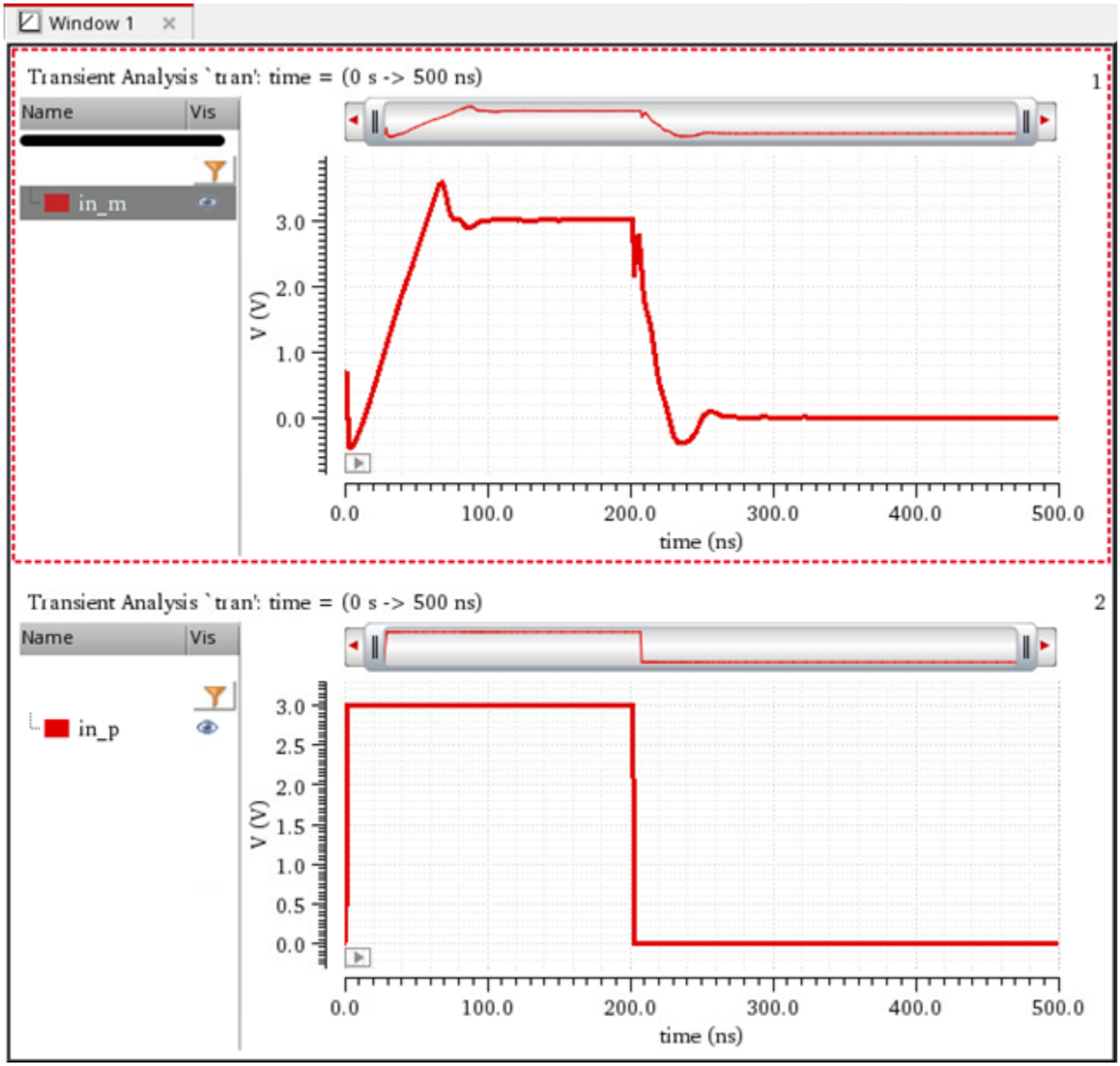
The following example plots the signal `in_m` from the `tran-tran` results of the specified results directory in subwindow `1` of the Waveform window you created using the function `awvCreatePlotWindow`.

```
awvPlotSignals('(("/servers/user/design/ampsim.raw" (("tran-tran" ("in_m"))))))
=> t
```

The following example plots the signal `in_p` from the `tran-tran` results of the specified results directory in subwindow `2` of the current Waveform window.

```
awvPlotSignals('(("/servers/user/design/ampsim.raw" (("tran-tran" ("in_p")))))
?plotStyle "New Subwindow")
```

```
=> t
```



The following example creates a waveform object `output` whose x and y axes are y axes of input waveforms `in_m` and `in_p` respectively.

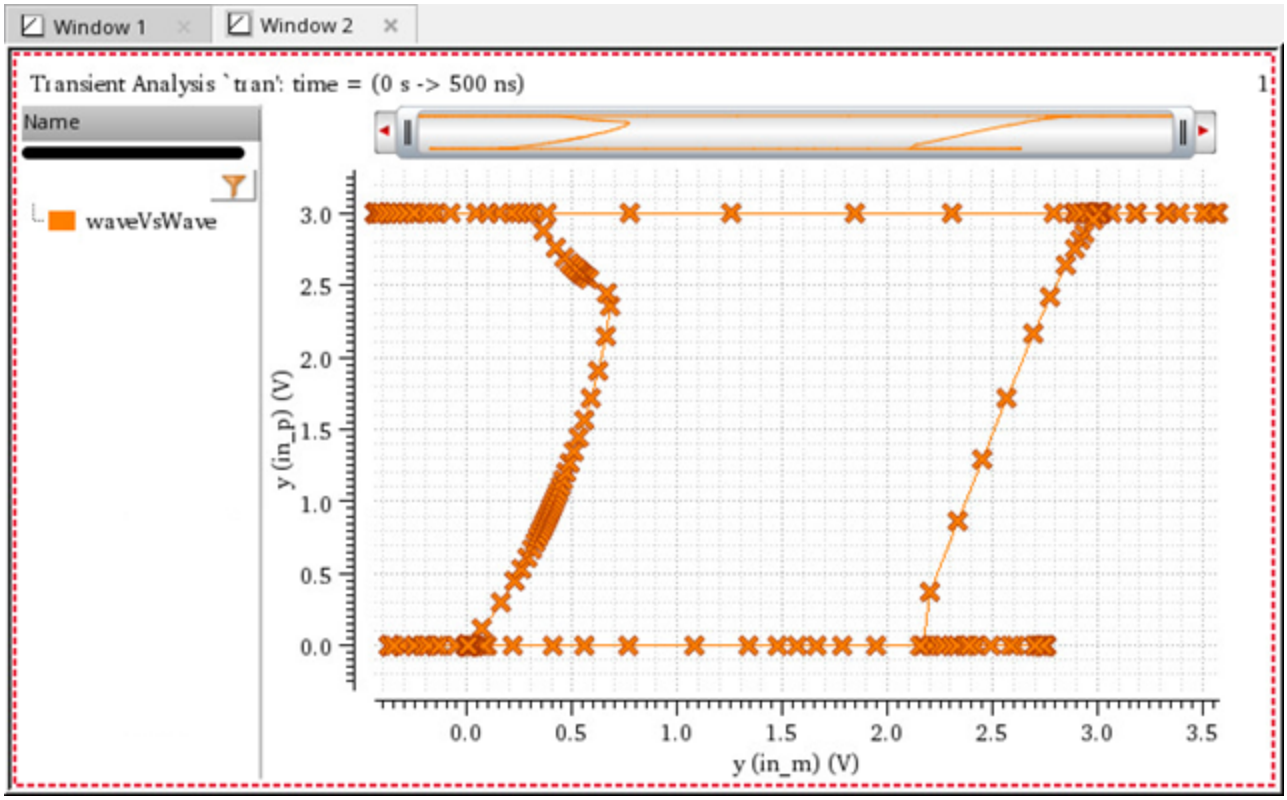
```
output=waveVsWave(?x v("in_m" ?result "tran-tran") ?y v("in_p" ?result "tran-
tran") ?xName "y (in_m)" ?xUnits "V" ?yName "y (in_p)" ?yUnits "V")
=> srrWave:0x36cde070
```

The following example creates another Waveform window and sets it as the current Waveform window.

```
awvCreatePlotWindow()
=> window:4
```

The following example plots the output waveform stored in the waveform object `output`.

```
awvPlotWaveform(
                awvGetCurrentWindow()
                list(output)
                ?expr list("waveVsWave")
                ?color list("y6")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("fine")
                ?showSymbols list(t)
                ?dataSymbol list("x")
                )
=> t
```



5

RF Functions

The RF SKILL functions let you perform various calculations on the results of RF analysis and plot their outputs.

Usually, these functions have the prefix `rf`.

This topic lists RF functions that are available in Virtuoso Visualization and Analysis XL.

rapidOIPN	rfCimMcpValue	rfEdgePhaseNoise
rfGetEventtimeIndex	rfGetMinDampFactor	rfInputNoise
rfJc	rfJcc	rfJitter
rfOutputNoise	rfThresholdXing	rfTotalPower
rfTransferFunction	rfWrlsCcdfValues	rfWrlsCim3Value
rfWrlsCim5Value	rfWrlsMeasContour	

rapidOIPN

```
rapidOIPN(
     t_result
     [ ?resultsDir t_resultsDirectory ]
     [ ?r n_resistance ]
     [ @rest l_args ]
     )
     => o_waveform / n_rapidOIPN / nil
```

Description

Calculates the value of nth-order output intercept point (OIPN) or returns the waveform of rapid OIPN.

Arguments

t_result	Name of the simulation result.
?resultsDir *t_resultsDirectory*	Path to the directory where simulation results are saved.
?r *n_resistance*	Value of resistance. Default value is `50` ohm.
@rest *l_args*	List of additional arguments that can be passed to the function.

Value Returned

o_waveform	Waveform representing an IPN curve.
n_rapidOIPN	Value of the output rapid IPN measurement.
`nil`	Indicates an error.

Examples

The following example opens simulation results of harmonic balance AC (`hbac`) analysis.

```
openResults("/servers/user/lib/cell/maestro/results/maestro/ExplorerRun.0/1/test/
psf")
=> "/servers/user/lib/cell/maestro/results/maestro/ExplorerRun.0/1/test/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
=> (hb_fi hb_fd hb_td hb_tran hbac_im3
    hbac_ip3 hbac model instance output
    designParamVals primitives subckts variables
)
```

The following example selects the result `hbac_ip3`.

```
selectResults('hbac_ip3)
=> stdobj@0x321174b8
```

The following example creates a waveform object that represents the power plotted on y axis against the design variable `prf`, which is sweeping from `-40` to `-20`.

```
wave1=rapidOIPN("hbac_ip3" ?r 50 '"prf" '-40)
=> srrWave:0x36730110
```

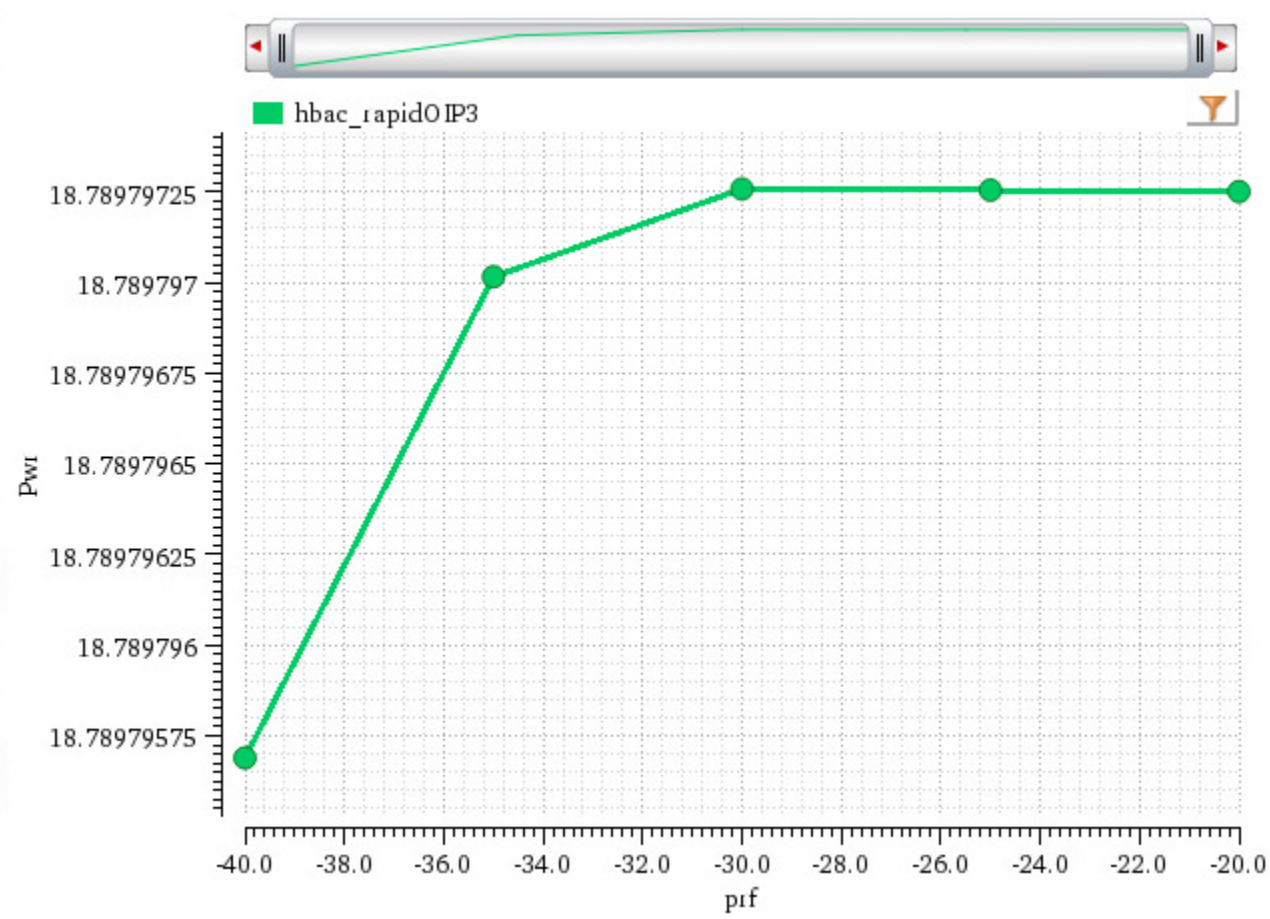
The following example creates a Waveform window `win1` and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example plots the waveform `wave1`, which represents rapid IP3 measurements, in the Waveform window `win1`.

```
awvPlotWaveform(
                win1
                list(wave1)
                ?expr list("hbac_rapidOIP3")
                ?color list("y18")
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list(".")
                )
```

```
=> t
```



The following example opens simulation results stored in the specified results directory.

```
openResults("/servers/user/lib/cell/maestro/results/maestro/ExplorerRun.0/1/test/
psf")
=> "/servers/user/lib/cell/maestro/results/maestro/ExplorerRun.0/1/test/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
ac(ac_im3 ac_ip3 hb_mt_fi hb_mt_fd pss_tran
    model instance output designParamVals primitives
    subckts variables
)
```

The following example selects the result `ac_ip3`.

```
selectResults('ac_ip3)
=> stdobj@0x3260b278
```

The following example calculates the output rapid IP3 value.

```
rapidOIPN("ac_ip3" ?r 50)
=> 16.46864
```

rfCimMcpValue

```
rfCimMcpValue(
     s_probe
     )
     => f_power / nil
```

Description

Returns the main channel power (MCP) value calculated for the specified probe when counter-intermodulation (CIM) is selected in LTE symbol.

Arguments

s_probe	Name of the probe for which the MCP value is to be calculated.

Value Returned

f_power	MCP value.
nil	MCP value cannot be calculated because of an error.

Examples

The following example opens simulation results of wireless envelope analysis stored in the specified results directory.

```
openResults("/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/Wireless/psf")
=> "/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/Wireless/psf"
```

The following examples return values of MCP calculated for the specified wireless probes `WPRB1` and `WPRB2`, respectively. These wireless probes are inserted into the input and output of the amplifier.

```
rfCimMcpValue("WPRB1")
=> -0.0364319
rfCimMcpValue("WPRB2")
=> 22.59707
```

rfEdgePhaseNoise

```
rfEdgePhaseNoise(
     [ ?result t_result ]
     [ ?eventList l_eventList ]
     [ ?resultsDir t_resultsDirectory ]
     )
     => 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*
: Name of the pnoise or hbnoise result.

`?eventList` *l_eventList*
: A list of two values, where the first value is the value of a sweep parameter and the second value is event time index.

  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.

  With sweeps, the values in this argument can be specified as:

  - `list(list())`
  - `'(list(`*val1 val2*`))`
  - `'(list(`*val1 val2*`) list(`*val3 val4*`) …)`

`?resultsDir` *t_resultsDirectory*
: Path to the results directory.

Value Returned

o_waveform
: Waveform representing instantaneous phase noise, conversion of jitter to phase noise, and spectrum plots related to jitter.

`nil`
: Indicates an error.

Examples

The following example opens simulation results of `hb-hbnoise` analysis stored in the specified results directory.

```
openResults("/home/user/lib/cell/view/results/maestro/ExplorerRun.0/1/test/psf")
=> "/home/user/lib/cell/view/results/maestro/ExplorerRun.0/1/test/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
=>
(envlp_td envlp_fd "envlp-fm.envlp" hb_fi hb_fd
    hb_td hbnoise_sample_hm0 hbnoise_sample_hm1 model instance
    output designParamVals primitives subckts variables
)
```

The following example selects the result `hbnoise_sample_hm0`.

```
selectResults('hbnoise_sample_hm0)
=> stdobj@0x31f7f788
```

The following example creates a waveform object wave that represents the edge phase noise.

```
wave=rfEdgePhaseNoise(?result "hbnoise_sample_hm0")
=> srrWave:0x36565070
```

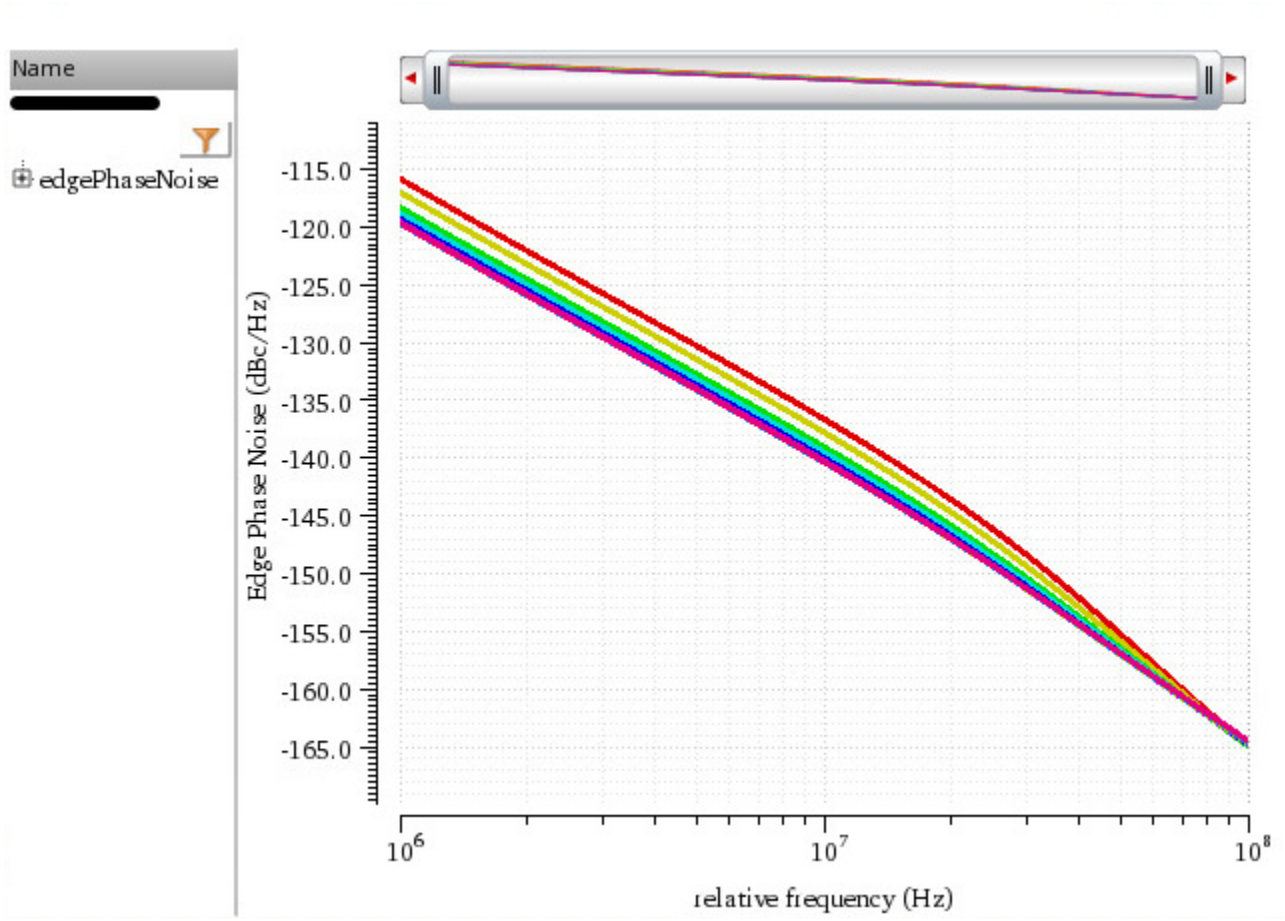
The following example creates a Waveform window and returns its window ID.

```
win=awvCreatePlotWindow()
=> window:3
```

The following example plots the waveform object `wave`, which represents the edge phase noise, in the Waveform window `win`.

```
awvPlotWaveform(
                win
                list(wave)
                ?expr list("edgePhaseNoise")
                )
```

```
=> t
```



The following example opens simulation results of pss-pnoise analysis stored in the specified directory.

```
openResults("/servers/user/testcase/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/pss_pnoise_trannoise/psf")
=> "/servers/user/testcase/lib/cell/maestro/results/maestro/ExplorerRun.0/1/
pss_pnoise_trannoise/psf"
```

The following example lists the available results in the currently open results directory.

```
results()
=>
(pss_tran pss_td pss_fd pnoise_sample_pm0 model
    instance output designParamVals primitives subckts
    variables
)
```

The following example selects the result `pnoise_sample_pm0`.

```
selectResults('pnoise_sample_pm0)
=> stdobj@0x314d0290
```

The following example creates a waveform object `wave1`, which represents the edge phase noise.

```
wave1=rfEdgePhaseNoise(?result "pnoise_sample_pm0")
=> srrWave:0x355ec0a0
```

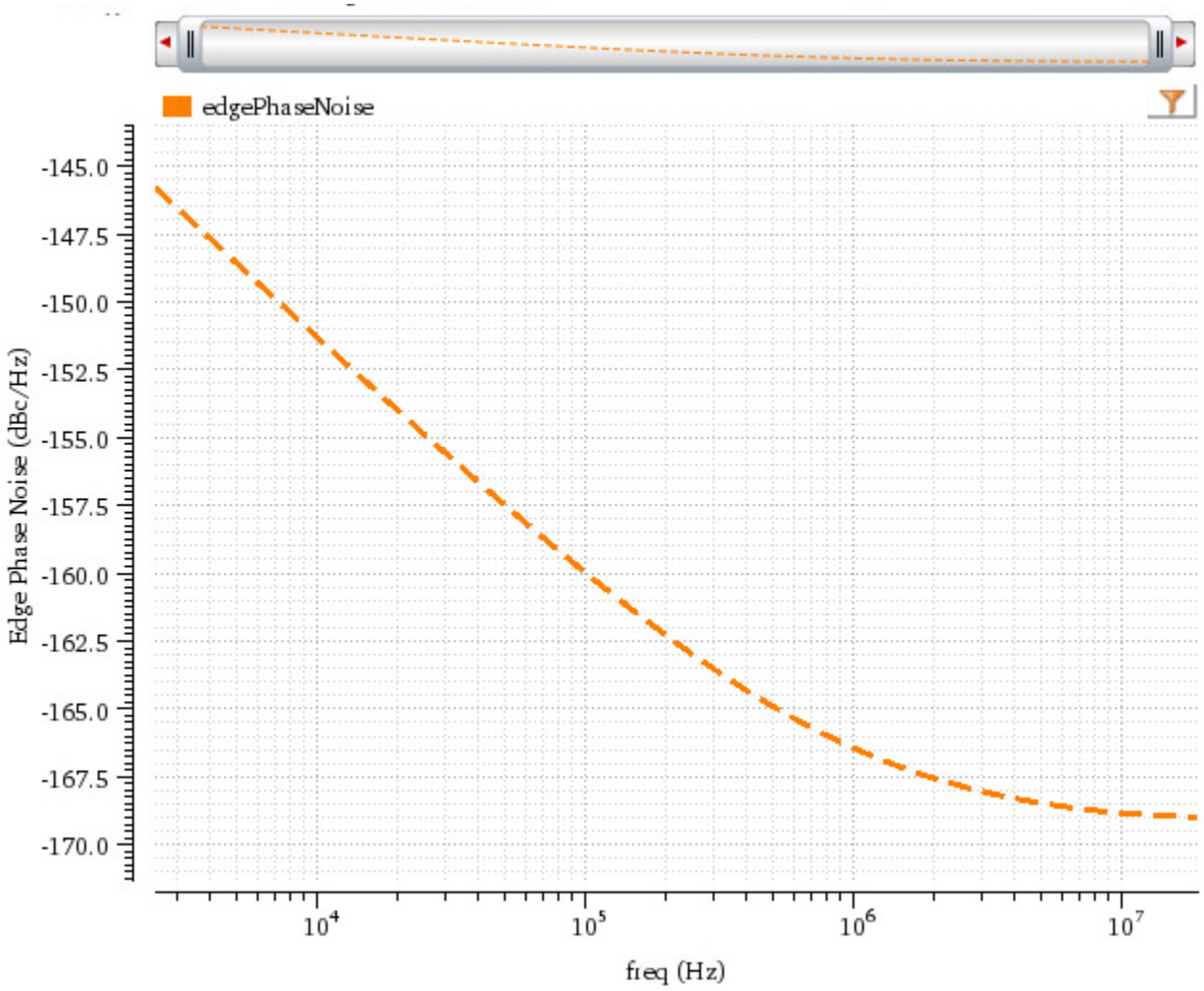
The following example creates a Waveform window and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example plots the waveform `wave1` in the Waveform window `win1`.

```
awvPlotWaveform(
                win1
                list(wave1)
                ?expr list("edgePhaseNoise")
                ?color list("y6")
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                )
```

```
=> t
```



rfGetEventtimeIndex

```
rfGetEventtimeIndex(
     t_signal
     t_resultName
     x_index
     )
     => n_eventIndex / 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.
x_index	Index value.

Value Returned

n_eventIndex	Event time of the signal for the specified index value.
nil	Event time cannot be returned because of an error.

Examples

The following example opens simulation results of periodic AC (pac) analysis stored in the specified results directory.

```
openResults("/servers/user/lib/cell/maestro/results/maestro/ExplorerRun.0/psf/
test/psf")
=> "/servers/user/lib/cell/maestro/results/maestro/ExplorerRun.0/psf/test/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
=> (pss_td pss_fd pac_sampled pac model
    instance output designParamVals primitives subckts
    variables
)
```

The following example selects the result pac_sampled.

```
selectResults('pac_sampled)
=> stdobj@0x322b7f80
```

The following example lists the outputs that are available in the selected result.

```
outputs()
("/RFin" "/RFout" "/vdd_rf!" "/gnd_rf!" "
    "/inp" "/inm" "/outp" "/outm" "
)
```

The following examples return event time values of the signal `RFout` for the index values `0` and `1`, respectively.

```
rfGetEventtimeIndex("RFout" "pac_sampled" 0)
=> 1.16671e-10
rfGetEventtimeIndex("RFout" "pac_sampled" 1)
=> 3.18086e-10
```

rfGetMinDampFactor

```
rfGetMinDampFactor(
     )
     => n_minDampFactor / nil
```

Description

Calculates the lowest damping factor or ratio for loops identified in the loop finder (`LF`) analysis.

Arguments

None

Value Returned

n_minDampFactor	Lowest damping factor.
`nil`	Lowest damping factor cannot be calculated because of an error.

Examples

The following example opens simulation results of loop finder analysis stored in the specified results directory.

```
openResults("/home/user/LoopFinder/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/test/psf")
=> "/home/user/LoopFinder/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/test/psf"
```

The following example lists the available results in the currently open results directory.

```
results()
=>
(pz stb stb_margin tran tranOp
    lf model instance output designParamVals
    primitives subckts variables
)
```

The following example selects results of the loop finder analysis `lf`.

```
selectResults('lf)
=> stdobj@0x31d962c0
```

The following example lists the outputs stored in the simulation results of loop finder analysis.

```
outputs()
=> ("/loop1")
```

The following example calculates the lowest damping factor for the loops identified in the loop finder analysis.

```
rfGetMinDampFactor()
=> 0.2594875
```

rfInputNoise

```
rfInputNoise(
     t_unit
     [ ?result t_noiseResultName ]
     )
     => o_waveform / nil
```

Description

Returns the input noise waveform. This function is run on the results of the Spectre `pss-pnoise` and `hb-hbnoise` analyses.

Arguments

t_unit	Specifies the unit of y axis. Valid values are: ■ `V/sqrt(Hz)` ■ `V**2/Hz` ■ `dBc/Hz` ■ `dBV/Hz`
`?result` *t_noiseResultName*	Name of the results file alias in which the input noise waveform is saved.

Value Returned

o_waveform	Output waveform representing the input noise.
`nil`	Input noise waveform cannot be created because of an error.

Examples

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"))
```

The following example opens simulation results of `hb-hbnoise` analysis stored in the specified results directory.

```
openResults("/servers/user/lib/cell/maestro/results/maestro/ExplorerRun.0/psf/
test/psf")
=> "/servers/user/lib/cell/maestro/results/maestro/ExplorerRun.0/psf/test/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
(hb_td hb_fd hb_fi hbac hbnoise_am
    hbnoise_am_xfersrc hbnoise_src hbnoise_pm hbnoise_pm_xfersrc hbnoise
    hbnoise_xfersrc hbnoise_lsb hbnoise_lsb_xfersrc model instance
    output designParamVals primitives subckts variables
)
```

The following example selects the result `hbnoise`.

```
selectResults('hbnoise)
=> stdobj@0x3227fa10
```

The following example creates a waveform object `wave1` that represents the plot for input noise with y-axis unit specified as `V/sqrt(Hz)`.

```
wave1=rfInputNoise("V/sqrt(Hz)" ?result "hbnoise")
=> srrWave:0x36862080
```

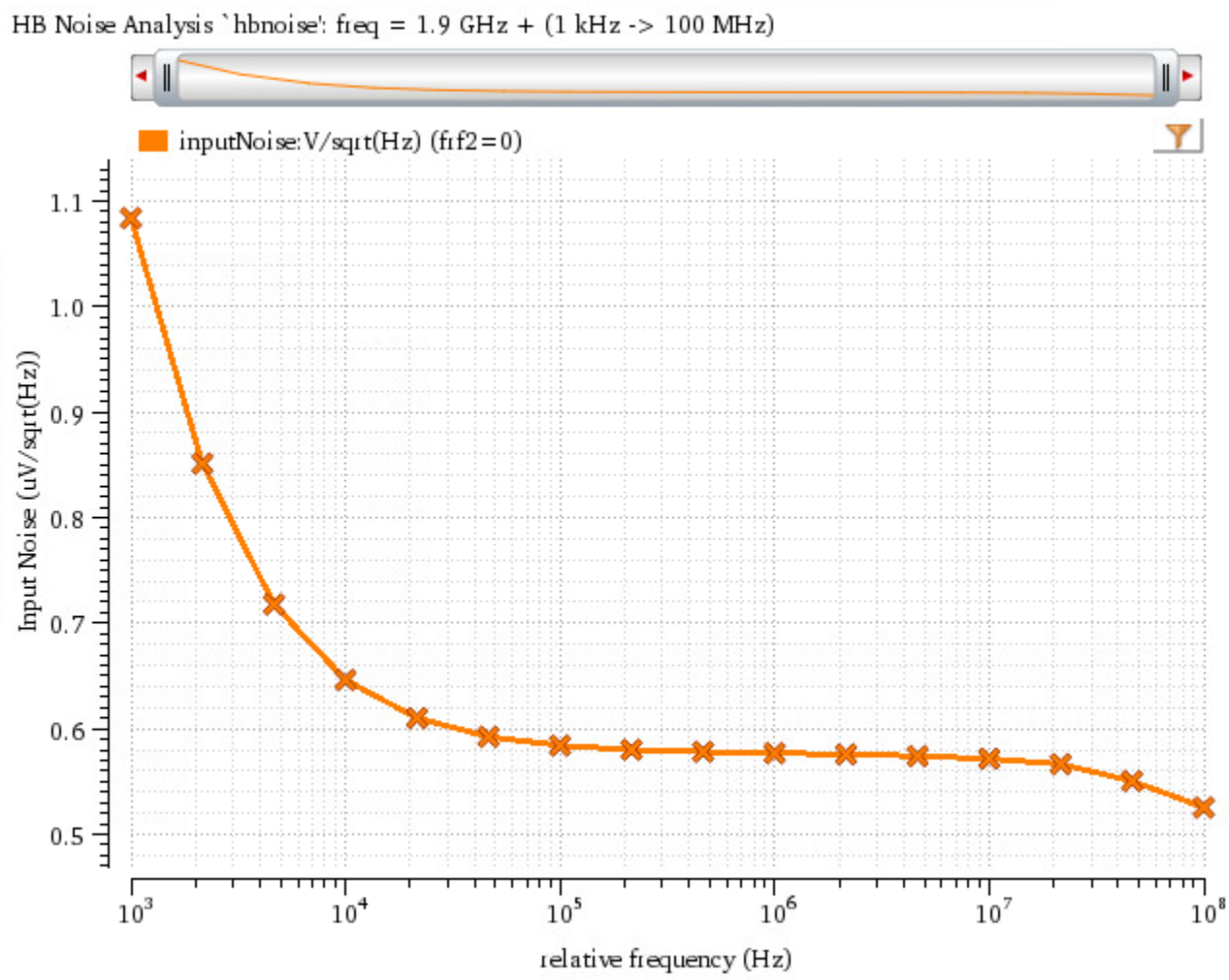
The following example creates a Waveform window `win1` and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example plots the waveform `wave1`, which represents the input noise, in the Waveform window `win1`.

```
awvPlotWaveform(
                win1
                list(wave1)
                ?expr list("inputNoise:V/sqrt(Hz)")
                ?color list("y6")
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("x")
                )
```

```
=> t
```



The following example creates a waveform object `wave1` that represents the plot for input noise with y-axis unit specified as `dBV/Hz`.

```
wave2=rfInputNoise("dBV/Hz" ?result "hbnoise")
=> srrWave:0x368621b0
```

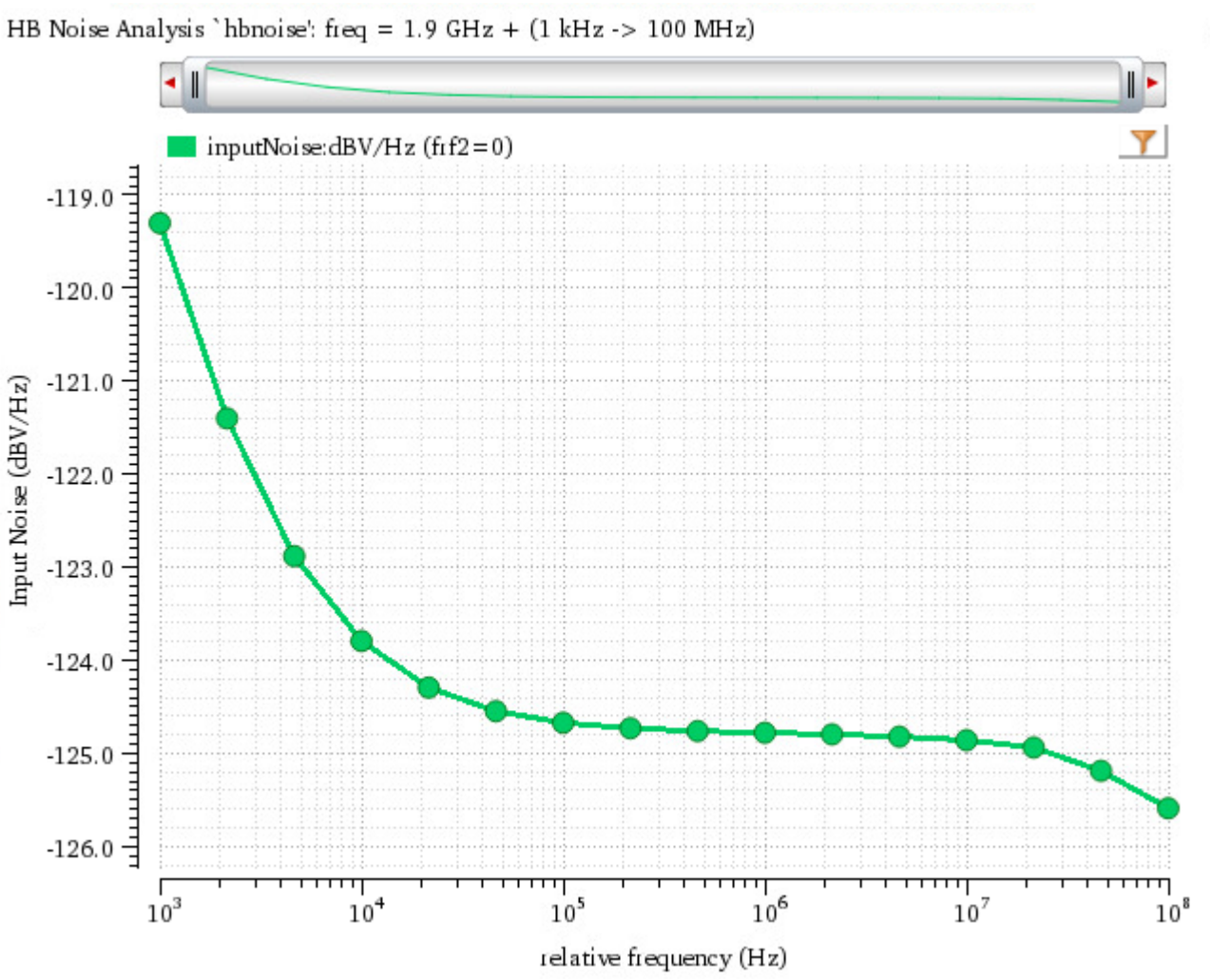
The following example creates another Waveform window `win2` and returns its window ID.

```
win2=awvCreatePlotWindow()
=> window:4
```

The following example plots the waveform `wave2`, which represents the input noise, in the Waveform window `win2`.

```
awvPlotWaveform(
                win2
```

```
                list(wave2)
                ?expr list("inputNoise:dBV/Hz")
                ?color list("y18")
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list(".")
                )
=> t
```



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 ]
     )
     => n_value / o_waveform / nil
```

Description

Calculates cycle jitter from the results of `hbnoise` or `pnoise` sample (jitter) analysis.

Arguments

`?result` *t_result*	Name of the result of Pnoise sample analysis.
`?resultsDir` *t_resultsDir*	Path to 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.
`?k` *n_k*	Number of cycles, which determines whether one period or *k*-periods jitter are calculated. The default value is `1`, which indicates that only one cycle jitter is calculated.
`?multiplier` *n_multiplier*	Frequency multiplier. The default value is `1`.

Value Returned

n_value	Value of cycle jitter if simulation is run in single run mode.
o_waveform	Waveform of cycle jitter if simulation is run in sweep mode.
nil	Result name is not correct or cycle jitter cannot be calculated because of an error.

Examples

The following example opens simulation results of `hbnoise` analysis stored in the specified results directory.

```
openResults("/home/user/hbnoise/lib/cell/view/results/maestro/ExplorerRun.0/1/
test/psf")
=> "/home/user/hbnoise/lib/cell/view/results/maestro/ExplorerRun.0/1/test/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
(hb_fi hb_fd hb_td hbnoise hbnoise_am
    hbnoise_pm hbnoise_lsb model instance output
    designParamVals primitives subckts variables
)
```

The following example selects the `hbnoise_pm` result from the current results directory.

```
selectResults('hbnoise_pm)
=> stdobj@0x315b06f8
```

The following example creates a waveform object `wave1` in which sweep variable is plotted on x axis and jitter value is plotted on y axis. The signal level is `rms`.

```
wave1=rfJc(?from 10K ?to 1G ?k 1 ?multiplier 1 ?result "hbnoise_pm" ?unit "Second")
=> srrWave:0x3574fbc0
```

The following example creates a Waveform window and returns its window ID.

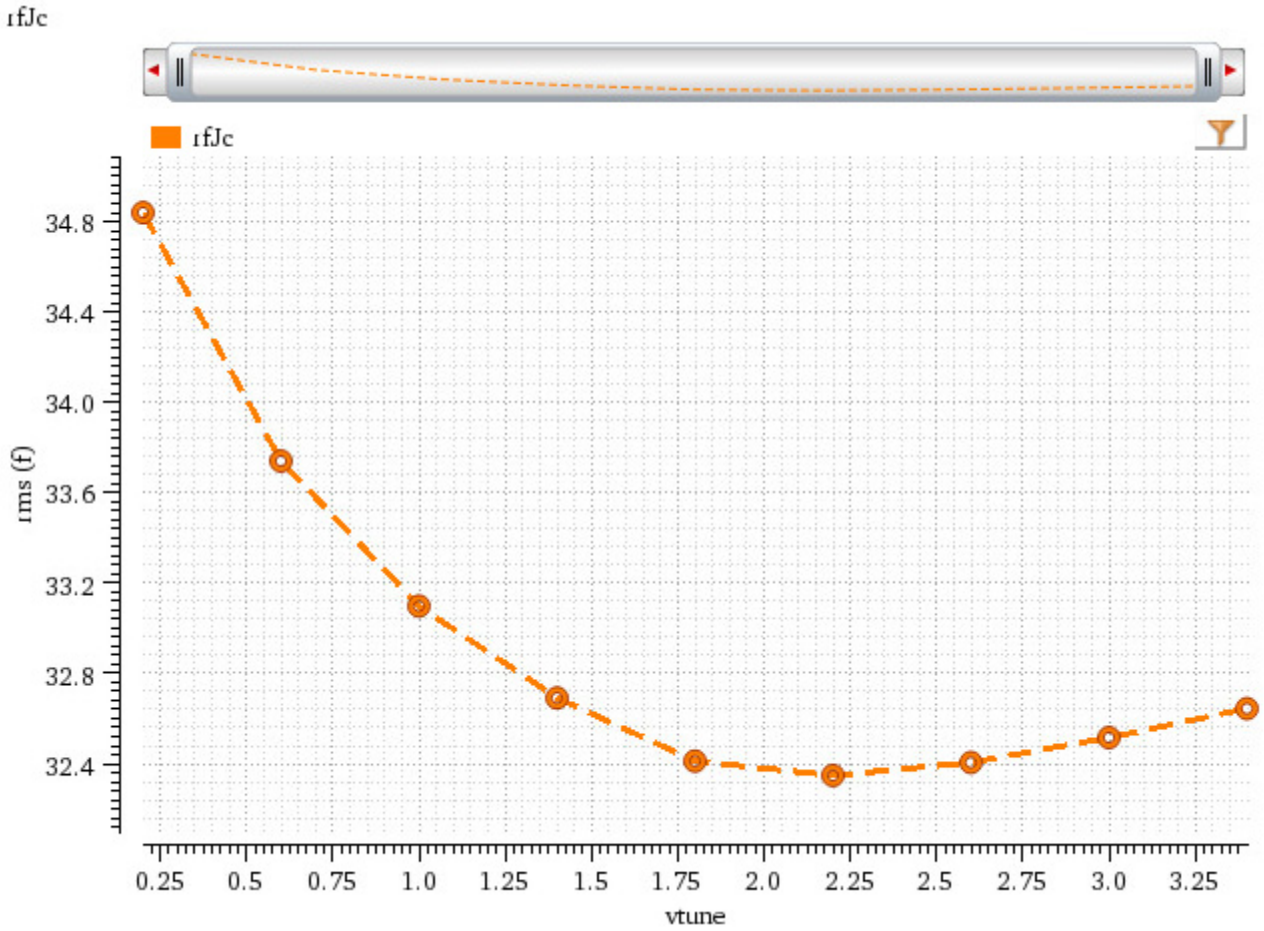
```
win1=awvCreatePlotWindow()
=> window:3
```

The following example plots the waveform `wave1` in the Waveform window `win1`.

```
awvPlotWaveform(
                win1
                list(wave1)
                ?expr list("rfJc")
```

```
                ?color list("y6")
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("o")
                )
=> t
```

Note that the sweep variable `vtune` is plotted on x axis and jitter values are plotted on y axis.



The following example creates a waveform object `wave2` in which sweep variable is plotted on x axis and jitter value is plotted on y axis. The signal level is peak-to-peak and BER is `1e-4`.

```
wave2=rfJc(?from 10K ?to 1G ?k 1 ?multiplier 1 ?result "hbnoise_pm" ?unit "Second"
?ber 1e-04)
=> srrWave:0x3574e7a0
```

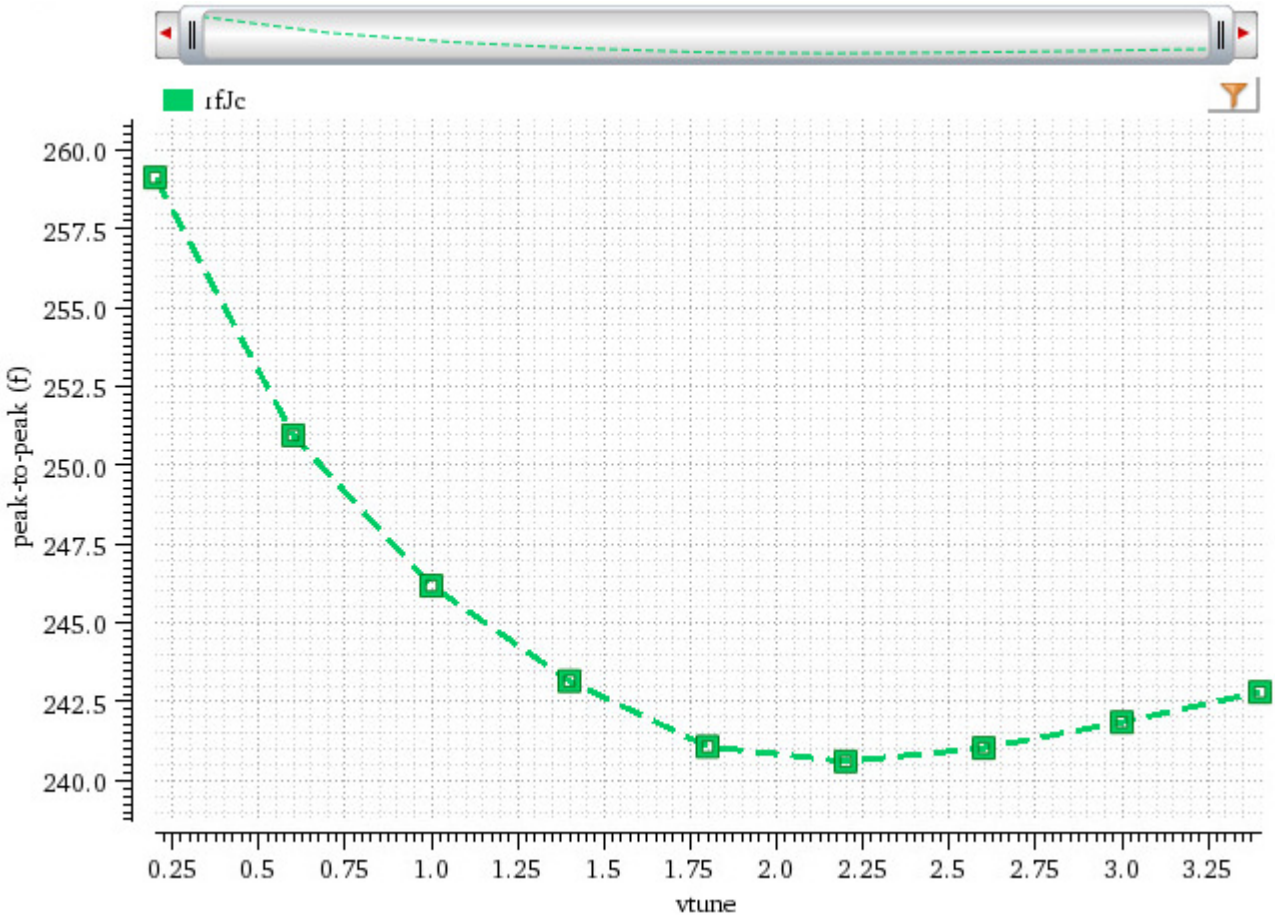
The following example creates a Waveform window and returns its window ID.

```
win2=awvCreatePlotWindow()
=> window:4
```

The following example plots the waveform `wave2` in the Waveform window `win2`.

```
awvPlotWaveform(
                win2
                list(wave2)
                ?expr list("rfJc")
                ?color list("y18")
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list(5)
                )
=> t
```

Note that the sweep variable `vtune` is plotted on x axis and jitter values are plotted on y axis.



The following example opens simulation results of `pss_pnoise` analysis stored in the specified results directory.

```
openResults("/servers/user/testcase/simulation/lib/cell/view/results/maestro/
ExplorerRun.0/1/pss_pnoise_trannoise/psf")
=> "/servers/user/testcase/simulation/lib/cell/view/results/maestro/
ExplorerRun.0/1/pss_pnoise_trannoise/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
(pss_tran pss_td pss_fd pnoise_sample_pm0 model
    instance output designParamVals primitives subckts
    variables
)
```

The following example selects the result `pnoise_sample_pm0` stored in the results directory.

```
selectResults('pnoise_sample_pm0)
=> stdobj@0x324df290
```

The following examples calculate values of cycle jitter from the result `pnoise_sample_pm0` in units `UI`, `Second`, and `ppm`, respectively. The signal level is `rms`.

```
rfJc(?from 2.5K ?to 19.2M ?k 1 ?multiplier 1 ?result "pnoise_sample_pm0" ?unit
"UI")
=> 5.02066e-06
rfJc(?from 2.5K ?to 19.2M ?k 1 ?multiplier 1 ?result "pnoise_sample_pm0" ?unit
"Second")
=> 1.307463e-13
rfJc(?from 2.5K ?to 19.2M ?k 1 ?multiplier 1 ?result "pnoise_sample_pm0" ?unit
"ppm")
=> 5.02066
```

The following examples calculate values of cycle jitter from the result `pnoise_sample_pm0` in units `UI`, `Second`, and `ppm`, respectively. The signal level is `peak-to-peak` and BER is `1e-12`.

```
rfJc(?from 2.5K ?to 19.2M ?k 1 ?multiplier 1 ?result "pnoise_sample_pm0" ?unit "UI"
?ber 1e-12)
=> 7.063566e-05
rfJc(?from 2.5K ?to 19.2M ?k 1 ?multiplier 1 ?result "pnoise_sample_pm0" ?unit
"Second" ?ber 1e-12)
=> 1.83947e-12
rfJc(?from 2.5K ?to 19.2M ?k 1 ?multiplier 1 ?result "pnoise_sample_pm0" ?unit
"ppm" ?ber 1e-12)
=> 70.63566
```

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 ]
     )
     => n_value / o_waveform / nil
```

Description

Calculates cycle-to-cycle jitter from the results of `hbnoise` or `pnoise` analysis.

Arguments

`?result` *t_result*	Name of the result of pnoise or hbnoise analysis.
`?resultsDir` *t_resultsDir*	Path to the results directory.
`?unit` *t_unit*	Unit of the 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. This argument can be used to convert between RMS and peak-to-peak random jitter. $Jitter_{Peak-to-peak} = \alpha \times Jitter_{RMS}$ Where α is the scaling factor. To know scaling factor (α) for various BER tolerance values, see RMS to Peak-to-Peak Jitter Conversion.
`?from` *n_from*	The lower frequency limiter.
`?to` *n_to*	The upper frequency limiter.

?k *n_k*	Number of cycles, which determines whether one period or *k*-periods jitter are calculated. The default value is 1, which indicates that only one cycle-to-cycle jitter is calculated.
?multiplier *n_multiplier*	Frequency multiplier. The default value is 1.

Value Returned

n_value	Value of cycle-to-cycle jitter if simulation is run in single run mode.
o_waveform	Waveform of cycle-to-cycle jitter if simulation is run in sweep mode.
nil	Result name is not correct or cycle-to-cycle jitter cannot be calculated because of an error.

Examples

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 x axis and cycle-to-cycle jitter value plotted on y axis if simulation is run in sweep mode.

```
rfJcc(?result "pnoise_sample_pm0" ?unit "Second" ?from 1000 ?to 10000 ?k 1
?multiplier 1)
```

The following example opens simulation results of hbnoise analysis stored in the specified results directory.

```
openResults("/home/user/hbnoise/lib/cell/view/results/maestro/ExplorerRun.0/1/
test/psf")
=> "/home/user/hbnoise/lib/cell/view/results/maestro/ExplorerRun.0/1/test/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
(hb_fi hb_fd hb_td hbnoise hbnoise_am
    hbnoise_pm hbnoise_lsb model instance output
    designParamVals primitives subckts variables
)
```

The following example selects the `hbnoise_pm` result from the current results directory.

```
selectResults('hbnoise_pm)
=> stdobj@0x315b06f8
```

The following example creates a waveform object `wave1` in which sweep variable is plotted on x axis and jitter value is plotted on y axis. The signal level is `rms`.

```
wave1=rfJcc(?from 10K ?to 1G ?k 1 ?multiplier 1 ?result "hbnoise_pm" ?unit
"Second")
=> srrWave:0x3574d370
```

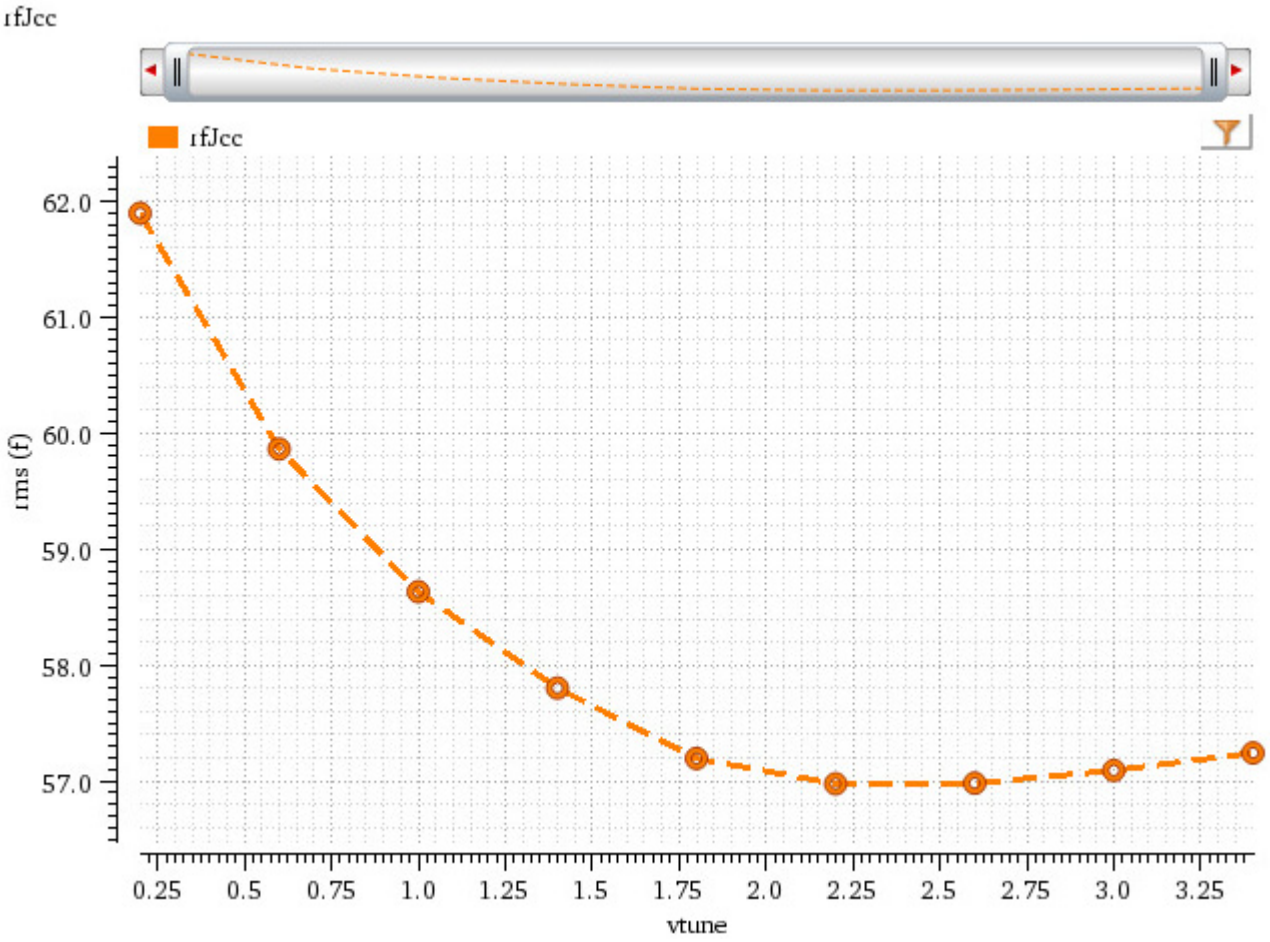
The following example creates a Waveform window and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example plots the waveform `wave1` in the Waveform window `win1`.

```
awvPlotWaveform(
                win1
                list(wave1)
                ?expr list("rfJcc")
                ?color list("y6")
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("o")
                )
=> t
```

Note that the sweep variable `vtune` is plotted on x axis and jitter values are plotted on y axis.



The following example creates a waveform object `wave2` in which sweep variable is plotted on x axis and jitter value is plotted on y axis. The signal level is peak-to-peak and BER is `1e-4`.

```
wave2=rfJcc(?from 10K ?to 1G ?k 1 ?multiplier 1 ?result "hbnoise_pm" ?unit "Second"
?ber 1e-04)
=> srrWave:0x3570c0c0
```

The following example creates a Waveform window and returns its window ID.

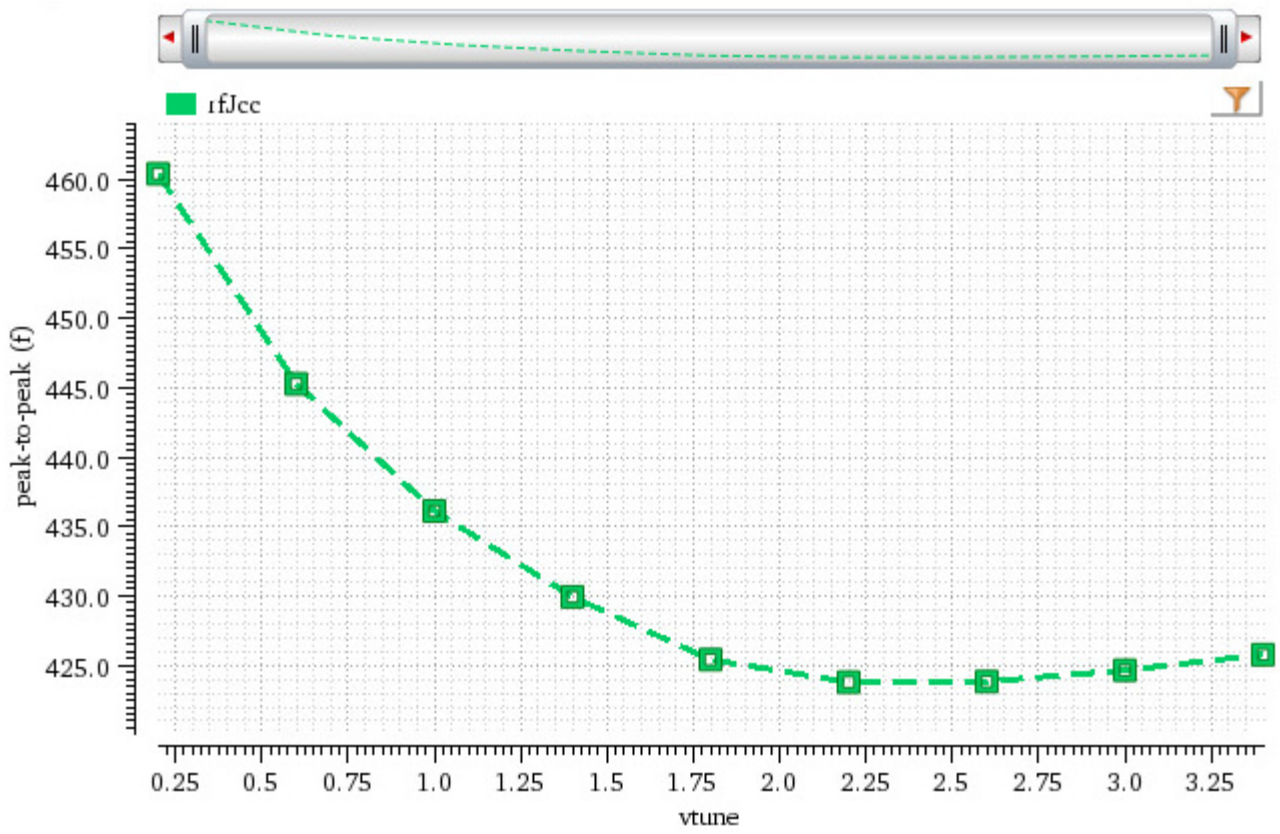
```
win2=awvCreatePlotWindow()
=> window:4
```

The following example plots the waveform `wave2` in the Waveform window `win2`.

```
awvPlotWaveform(
```

```
                win2
                list(wave2)
                ?expr list("rfJcc")
                ?color list("y18")
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("5")
                )
=> t
```

Note that the sweep variable `vtune` is plotted on x axis and jitter values are plotted on y axis.



The following example opens simulation results of `pss_pnoise` analysis stored in the specified results directory.

```
openResults("/servers/user/testcase/simulation/lib/cell/view/results/maestro/
ExplorerRun.0/1/pss_pnoise_trannoise/psf")
```

```
=> "/servers/user/testcase/simulation/lib/cell/view/results/maestro/
ExplorerRun.0/1/pss_pnoise_trannoise/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
(pss_tran pss_td pss_fd pnoise_sample_pm0 model
    instance output designParamVals primitives subckts
    variables
)
```

The following example selects the result `pnoise_sample_pm0` stored in the results directory.

```
selectResults('pnoise_sample_pm0)
=> stdobj@0x324df290
```

The following examples calculate values of cycle-to-cycle jitter from the result `pnoise_sample_pm0` in units `UI`, `Second`, and `ppm`, respectively. The signal level is `rms`.

```
rfJcc(?from 2.5K ?to 19.2M ?k 1 ?multiplier 1 ?result "pnoise_sample_pm0" ?unit
"UI")
=> 8.647249e-06
rfJcc(?from 2.5K ?to 19.2M ?k 1 ?multiplier 1 ?result "pnoise_sample_pm0" ?unit
"Second")
=> 2.251888e-13
rfJcc(?from 2.5K ?to 19.2M ?k 1 ?multiplier 1 ?result "pnoise_sample_pm0" ?unit
"ppm")
=> 8.647249
```

The following examples calculate values of cycle-to-cycle jitter from the result `pnoise_sample_pm0` in units `UI`, `Second`, and `ppm`, respectively. The signal level is `peak-to-peak` and BER is `1e-12`.

```
rfJcc(?from 2.5K ?to 19.2M ?k 1 ?multiplier 1 ?result "pnoise_sample_pm0" ?unit
"UI" ?ber 1e-12)
=> 0.0001216581
rfJcc(?from 2.5K ?to 19.2M ?k 1 ?multiplier 1 ?result "pnoise_sample_pm0" ?unit
"Second" ?ber 1e-12)
=> 3.168181e-12
rfJcc(?from 2.5K ?to 19.2M ?k 1 ?multiplier 1 ?result "pnoise_sample_pm0" ?unit
"ppm" ?ber 1e-12)
=> 121.6581
```

RMS to Peak-to-Peak Jitter Conversion

To convert between RMS and peak-to-peak random jitter, the argument `?ber` must be specified. The following equation can be used to convert between the two:

$$Jitter_{Peak-to-peak} = \alpha \times Jitter_{RMS}$$

The following table list the scaling factor (α) for various BER tolerance values.

BER	Scaling Factor (Alpha)
10^{-3}	6.180
10^{-4}	7.438
10^{-5}	8.530
10^{-6}	9.507
10^{-7}	10.399
10^{-8}	11.224
10^{-9}	11.996
10^{-10}	12.723
10^{-11}	13.412
10^{-12}	14.069
10^{-13}	14.698
10^{-14}	15.301
10^{-15}	15.883
10^{-16}	16.444

and

rfJitter

```
rfJitter(
     [ ?result t_result ]
     [ ?resultsDir t_resultsDir ]
     [ ?unit t_unit ]
     [ ?ber g_ber ]
     [ ?from n_from ]
     [ ?to n_to ]
     [ ?signalLevel t_signalLevel ]
     )
     => n_value / o_waveform / nil
```

Description

Calculates jitter from the results of `hbnoise` and `pnoise` sample (jitter) analyses. It is used to calculate `Jee`, `JDelay`, and `RMS Phase Noise`.

Arguments

`?result` *t_result*	Name of the result of `hbnoise` or `pnoise` sample analysis.
`?resultsDir` *t_resultsDir*	Path to 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`.

Value Returned

n_value	Jitter value if simulation is run in single run mode.
o_waveform	Waveform of jitter if simulation is run in sweep mode.

`nil`	Result name is not correct and jitter cannot be calculated.

Examples

The following example opens simulation results of `hb-hbnoise` analysis stored in the specified results directory.

```
openResults("/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/psf/test/psf")
=> "/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/psf/test/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
=>
tran(tranOp dcOp dcOpInfo dc hb_fi
    hb_fd hb_td hbnoise_sample_hm0 hbnoise_sample_hm1 model
    instance output designParamVals primitives subckts
    variables
)
```

The following example selects the result `hbnoise_sample_hm0`.

```
selectResults('hbnoise_sample_hm0)
=> stdobj@0x312f8a70
```

The following example creates a waveform object `wave1`, representing the jitter waveform. The signal level is `rms`.

```
wave1=rfJitter(?result "hbnoise_sample_hm0" ?unit "Second" ?from 10K ?to 1G
?signalLevel "rms")
=> srrWave:0x354f6060
```

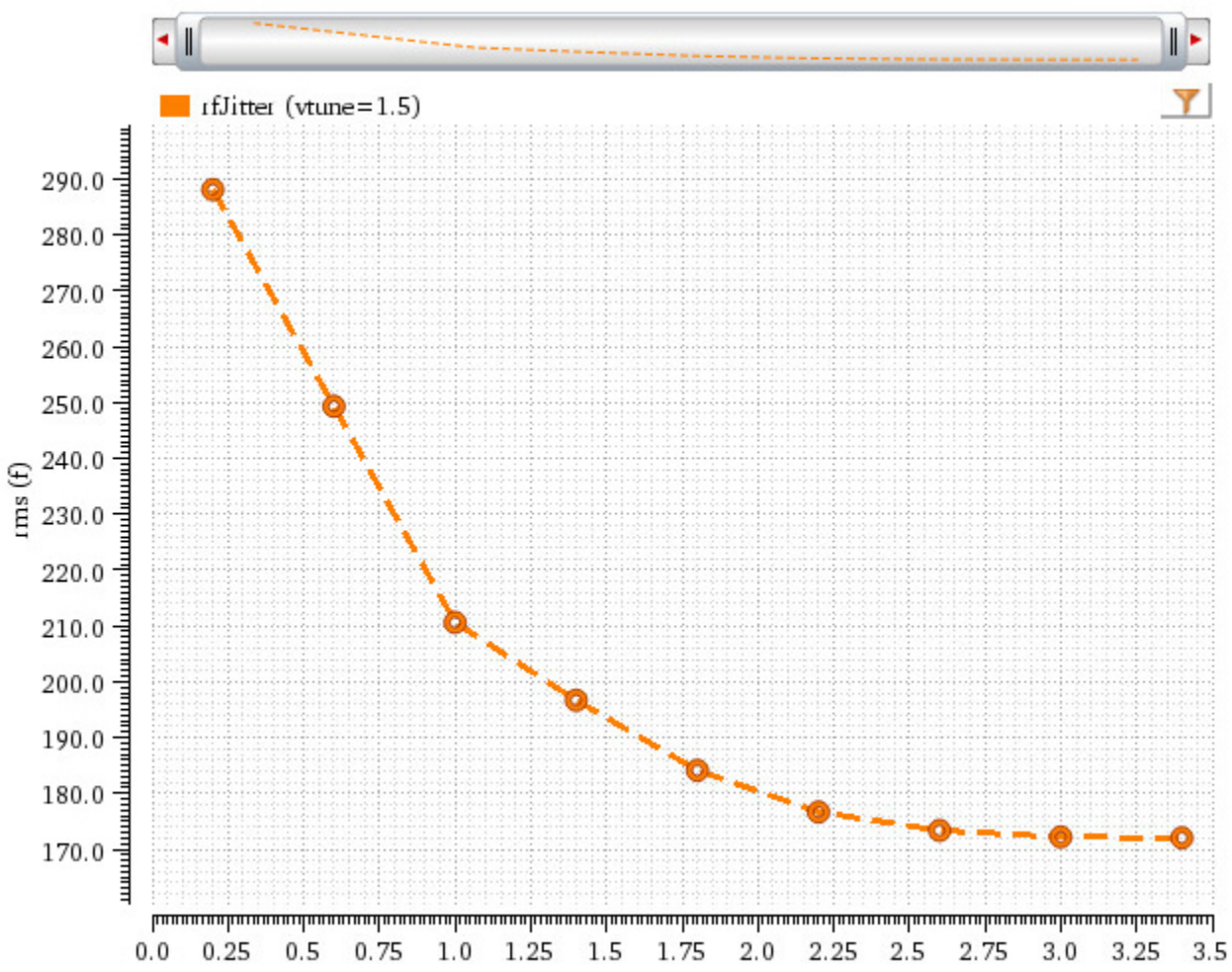
The following example creates a Waveform window `win1` and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example plots the jitter waveform in the window `win1`.

```
awvPlotWaveform(
                win1
                list(wave1)
                ?expr list("rfJitter")
                ?color list("y6")
                ?lineType list("line")
                ?lineStyle list("dash")
```

```
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("o")
                )
=> t
```



The following example creates a waveform object `wave2`, representing the jitter waveform. The signal level is `peak-to-peak` and BER is `1e-4`.

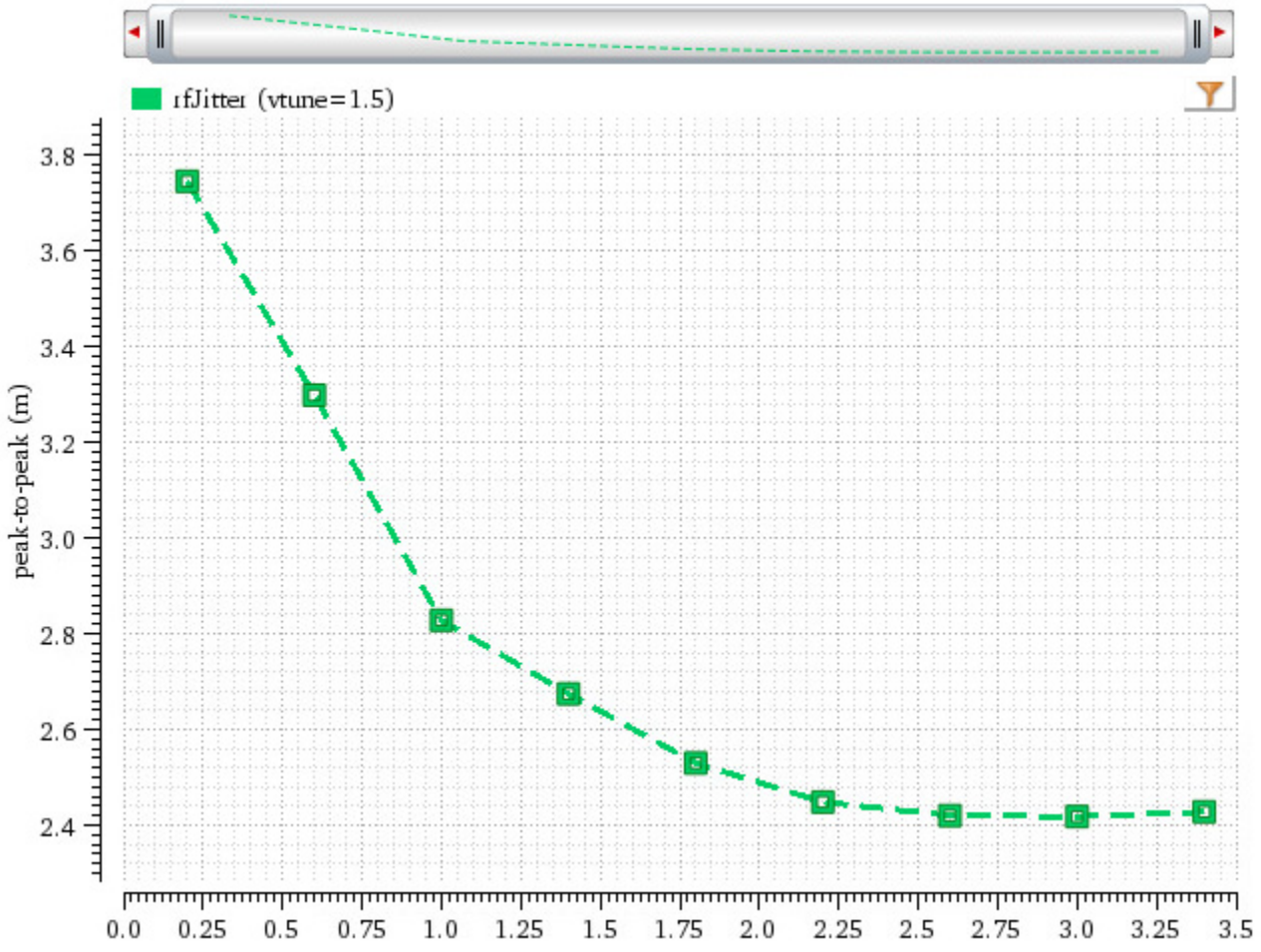
```
wave2=rfJitter(?result "hbnoise_sample_hm0" ?unit "UI" ?from 10K ?to 1G
?signalLevel "peak-to-peak" ?ber 1e-4)
=> srrWave:0x354f5a00
```

The following example creates another Waveform window `win2` and returns its window ID.

```
win2=awvCreatePlotWindow()
=> window:4
```

The following example plots the jitter waveform in the window `win2`.

```
awvPlotWaveform(
                win2
                list(wave2)
                ?expr list("rfJitter")
                ?color list("y18")
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list(5)
                )
=> t
```



The following example opens simulation results of `pss_pnoise` analysis stored in the specified results directory.

```
openResults("/servers/user/testcase/simulation/lib/cell/view/results/maestro/
ExplorerRun.0/1/pss_pnoise_trannoise/psf")
=> "/servers/user/testcase/simulation/lib/cell/view/results/maestro/
ExplorerRun.0/1/pss_pnoise_trannoise/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
(pss_tran pss_td pss_fd pnoise_sample_pm0 model
    instance output designParamVals primitives subckts
    variables
)
```

The following example selects the result pnoise_sample_pm0 stored in the results directory.

```
selectResults('pnoise_sample_pm0)
=> stdobj@0x324df290
```

The following example calculates the jitter value from the result `pnoise_sample_pm0`. The start frequency is `2.5K` and stop frequency is `19.2M`. The signal level is `rms`.

```
rfJitter(?result "pnoise_sample_pm0" ?unit "Second" ?from 2.5K ?to 19.2M
?signalLevel "rms")
=> 1.039041e-13
```

The following example calculates the jitter value from the result `pnoise_sample_pm0`. The start frequency is `2.5K` and stop frequency is `19.2M`. The signal level is `peak-to-peak` and BER is `1e-12`.

```
rfJitter(?result "pnoise_sample_pm0" ?unit "Second" ?from 2.5K ?to 19.2M
?signalLevel "peak-to-peak" ?ber 1e-12)
=> 1.461827e-12
```

rfOutputNoise

```
rfOutputNoise(
     t_unit
     [ ?result t_noiseResultName ]
     [ ?noiseConvention t_noiseConventionType ]
     )
     => o_waveform / nil
```

Description

Returns the output noise waveform. This function is run on the results of the Spectre `pss-pnoise` and `hb-hbnoise` analyses.

Arguments

t_unit Specifies the y-axis unit.

Valid values are:

- `V/sqrt(Hz)`
- `V**2/Hz`
- `dBc/Hz`
- `dBV/Hz`

`?result` *t_noiseResultName*

Name of the results of `pss-pnoise` or `hb-hbnoise` analysis.

`?noiseConvention` *t_noiseConventionType*

Specifies the type of noise convention.

The valid values are:

- `SSB`: Single-sideband.
- `DSB`: Double-sideband.

The default value is `SSB`.

This argument is used for the noise type `AM` or `PM`.

Value Returned

o_waveform	Waveform object representing the output noise.
nil	Output noise cannot be calculated because of an error.

Examples

The following example opens simulation results stored in the specified directory.

```
openResults("/home/user/hbnoise/ExampleLibRF/oscillator_ckt/maestro/results/
maestro/ExplorerRun.0/1/ExampleLibRF_oscillator_ckt_1/psf")
=> "/home/user/hbnoise/ExampleLibRF/oscillator_ckt/maestro/results/maestro/
ExplorerRun.0/1/ExampleLibRF_oscillator_ckt_1/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
(hb_fi hb_fd hb_td hbnoise hbnoise_am
    hbnoise_pm hbnoise_lsb model instance output
    designParamVals primitives subckts variables
)
```

The following example creates a waveform object `wave1` that represents the waveform of output noise from the result `hbnoise_am`, which is available in simulation results of `hb-hbnoise` analysis. The y-axis unit of the waveform is `dBc/Hz` and noise convention is `DSB`.

```
wave1=rfOutputNoise("dBc/Hz" ?result "hbnoise_am" ?noiseConvention "DSB")
=> srrWave:0x36647920
```

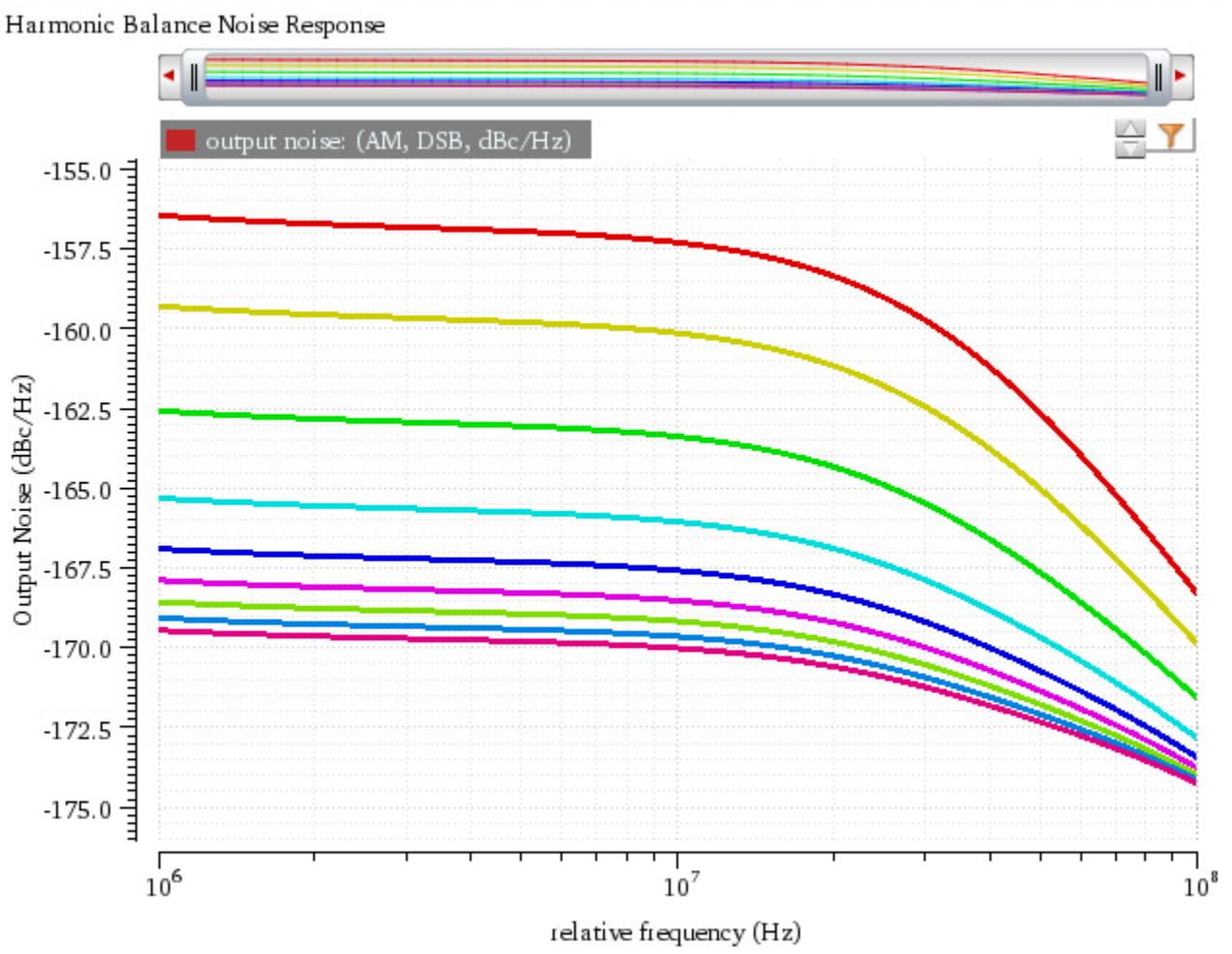
The following example creates a Waveform window and returns its widow ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example plots the waveform object `wave1` in the Waveform window `win1`. Note that the unit of y axis is `dBc/Hz`.

```
awvPlotWaveform(
                win1
                list(wave1)
                ?expr list("output noise: (AM, DSB, dBc/Hz)")
                )
```

```
=> t
```



The following example creates a waveform object `wave2` that represents the waveform of output noise from the result `hbnoise_pm`, which is available in simulation results of `hb-hbnoise` analysis. The y-axis unit of the waveform is `V/sqrt(Hz)` and noise convention is `DSB`.

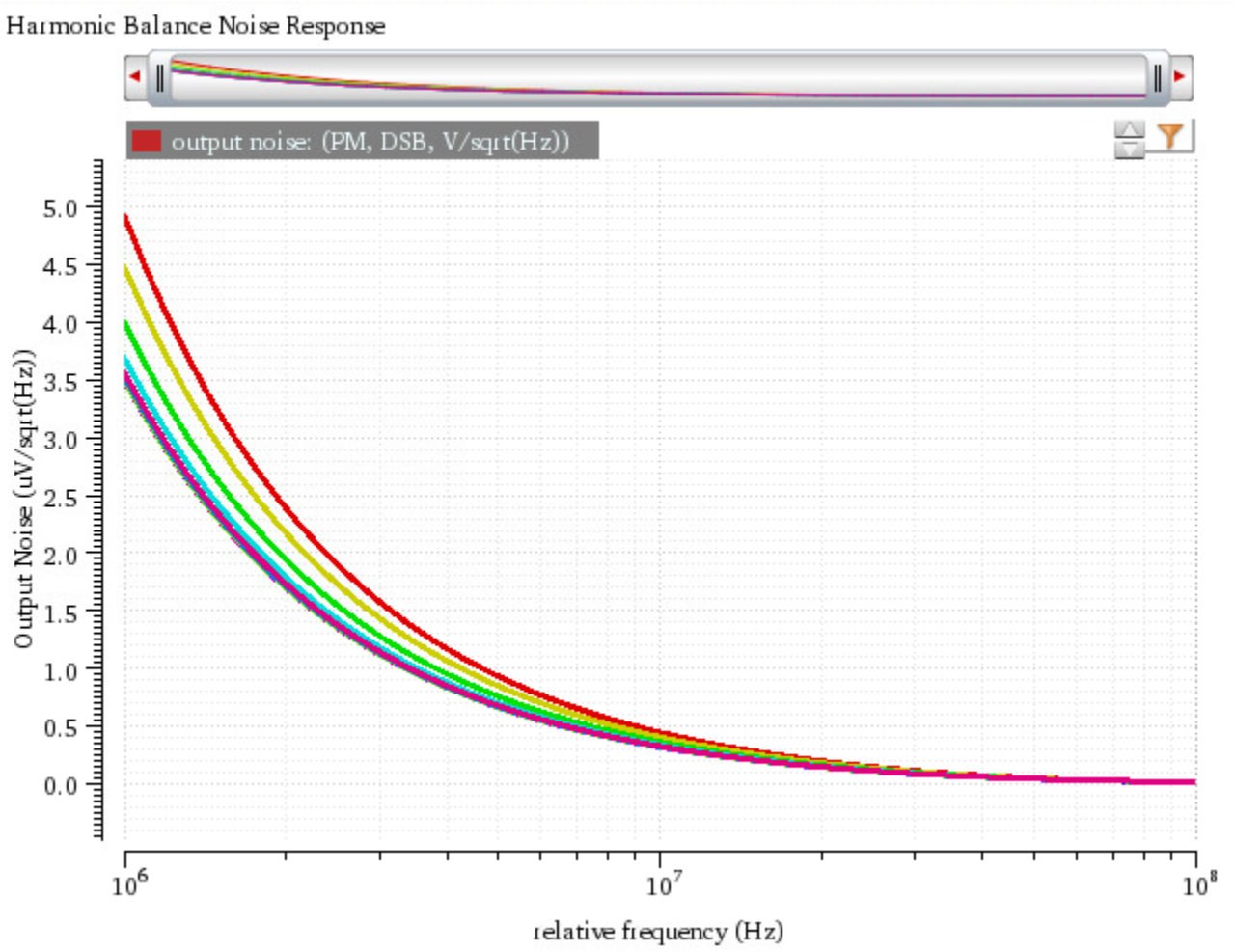
```
wave2=rfOutputNoise("V/sqrt(Hz)" ?result "hbnoise_pm" ?noiseConvention "DSB")
=> srrWave:0x36646040
```

The following example creates a Waveform window and returns its widow ID.

```
win2=awvCreatePlotWindow()
window:4
```

The following example plots the waveform object wave1 in the Waveform window `win2`. Note that the unit of y axis is `V/sqrt(Hz)`.

```
awvPlotWaveform(
                win2
                list(wave2)
                ?expr list("output noise: (PM, DSB, V/sqrt(Hz))")
                )
=> t
```



rfThresholdXing

```
rfThresholdXing(
     [ ?result t_result ]
     [ ?resultsDir t_resultsDirectory ]
     )
     => o_waveform / nil
```

Description

Calculates the threshold crossing value according to the jitter event time from the results of `pnoise` or `hbnoise` sample (jitter) analysis.

Arguments

`?result` *t_result*	Name of the result of `pnoise` or `hbnoise` sample analysis.
`?resultsDir` *t_resultsDirectory*	Path to the results directory where simulation results of `pnoise` or `hbnoise` sample analysis are saved.

Value Returned

o_waveform	Waveform of threshold crossing with a time range, and a marker on the waveform with the threshold value.
`nil`	Threshold crossing value cannot be calculated because the specified result name is not correct.

Examples

The following example opens simulation results of `pnoise` sample analysis stored in the results directory.

```
openResults("/servers/user/rfJitter/testcase/simulation/Lib/Cell/View/results/
maestro/ExplorerRun.0/1/pss_pnoise_trannoise/psf")
=> "/servers/user/rfJitter/testcase/simulation/Lib/Cell/View/results/maestro/
ExplorerRun.0/1/pss_pnoise_trannoise/psf"
```

The following example lists the results stored in the currently open results directory.

```
results()
(pss_tran pss_td pss_fd pnoise_sample_pm0 model
```

```
    instance output designParamVals primitives subckts
    variables
)
```

The following example creates a waveform object `wave1` that represents the waveform of threshold crossing values plotted against a time range. The waveform is plotted from the `pnoise_sample_pm0` result.

```
wave1=rfThresholdXing(?result "pnoise_sample_pm0")
=> srrWave:0x35d19050
```

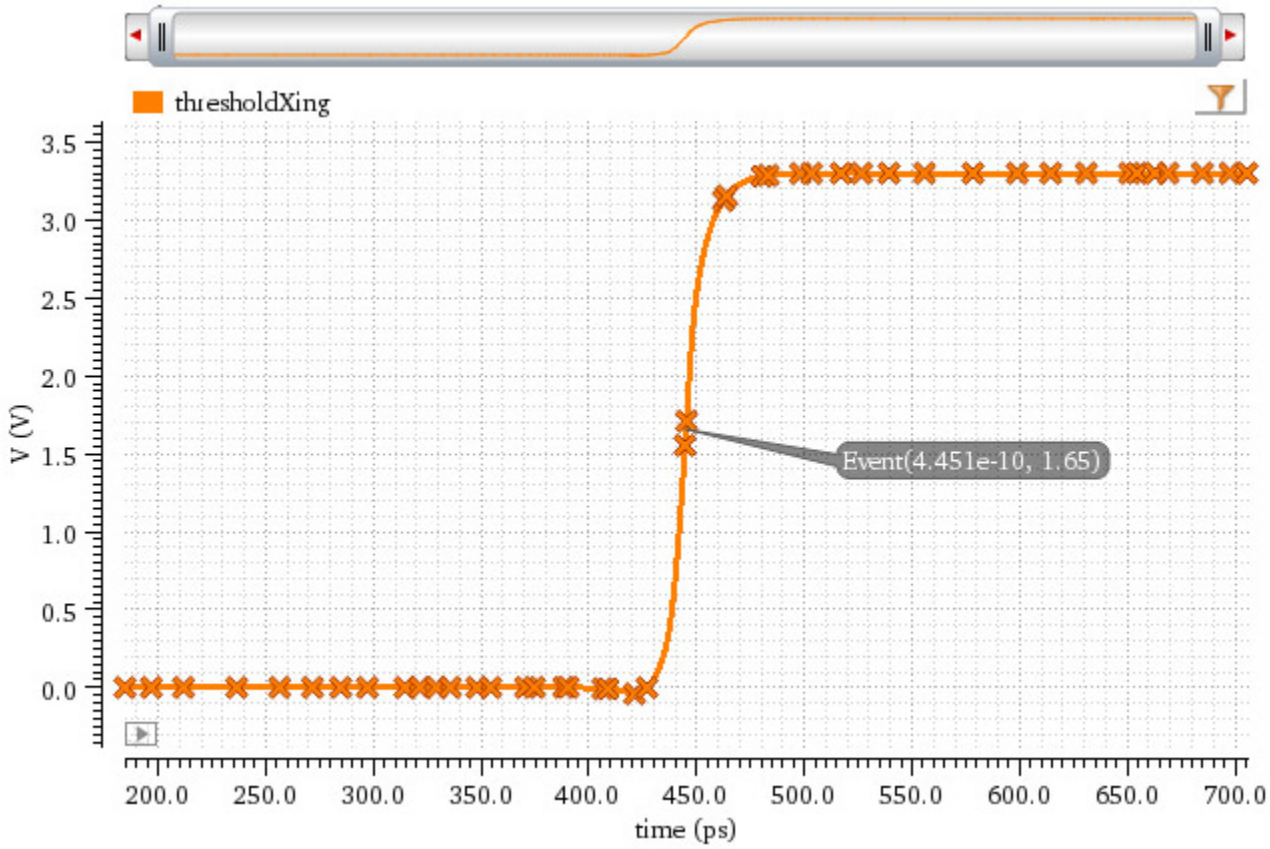
The following example creates a Waveform window and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example plots the waveform object `wave1` in the Waveform window `win1`.

```
awvPlotWaveform(
                win1
                list(wave1)
                ?expr list("thresholdXing")
                ?color list("y6")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("X")
                )
=> t
```

Note that the time is plotted on x axis and voltage is plotted on y axis. The marker on the waveform shows the time and threshold value.



rfTotalPower

```
rfTotalPower(
     o_currentWave
     o_voltageWave
     x_harmonic
     )
     => f_powerReturn / nil
```

Description

Calculates the root mean square (RMS) power between two terminals for fast and regular envelop analysis.

Arguments

o_currentWave	Waveform object that represents a current signal.
o_voltageWave	Waveform object that represents a voltage signal.
x_harmonic	Number of harmonics.

Value Returned

f_powerReturn	RMS power between the two terminals.
nil	RMS power cannot be calculated because of an error.

Examples

The following example opens simulation results of envelope analysis stored in the following results directory.

```
openResults("/servers/user/wireless/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/Wireless/psf")
=> "/servers/user/wireless/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/Wireless/psf"
```

The following example lists results available in the currently open results directory.

```
results()
=> (envlp_fd conste binary fwrls ccdf
    avgwrls param model instance output
    designParamVals primitives subckts variables
)
```

The following example opens the result `envlp_fd` from the current results directory.

```
selectResults('envlp_fd)
=> stdobj@0x3219a248
```

The following example lists the outputs available in the selected result.

```
outputs()
=> ("/net02" "/net2" "/V5/PLUS" "/V9/PLUS" "/iSignal")
```

The following example creates a waveform object for the current signal `iSignal`.

```
i=i("iSignal")
=> srrWave:0x362c2020
```

The following example creates a waveform object v for the voltage signal `net02`.

```
v=v("net02")
=> srrWave:0x362c2030
```

The following example calculates the total RMS power.

```
rfTotalPower(i v 1)
=> 0.1871363
```

rfTransferFunction

```
rfTransferFunction(
     t_unit
     [ ?result t_resultName ]
     )
     => o_waveform / nil
```

Description

Returns the waveform representing the transfer function. This function is run on the results of the pss-pnoise and hb-hbnoise analysis.

Arguments

t_unit	Specifies the unit of y-axis. Valid values are: ■ V/V ■ dB
?result *t_noiseResultName*	Name of the results file alias in which the gain waveform is saved.

Value Returned

o_waveform	Waveform representing the transfer function.
nil	Indicates an error.

Examples

The following example opens simulation results of `hb-hbnoise` analysis stored in the specified directory.

```
openResults("/servers/user/lib/cell/maestro/results/maestro/ExplorerRun.0/psf/
test/psf")
=> "/servers/user/lib/cell/maestro/results/maestro/ExplorerRun.0/psf/test/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
```

```
=>
(hb_td hb_fd hb_fi hbac hbnoise_am
    hbnoise_am_xfersrc hbnoise_src hbnoise_pm hbnoise_pm_xfersrc hbnoise
    hbnoise_xfersrc hbnoise_lsb hbnoise_lsb_xfersrc model instance
    output designParamVals primitives subckts variables
)
```

The following example selects the `hbnoise` result.

```
selectResults('hbnoise)
=> stdobj@0x31a12a10
```

The following example creates a waveform object `wave1` that represents the plot for transfer function with y-axis unit specified as `V/V`.

```
wave1=rfTransferFunction("V/V" ?result "hbnoise")
=> srrWave:0x35ff7030
```

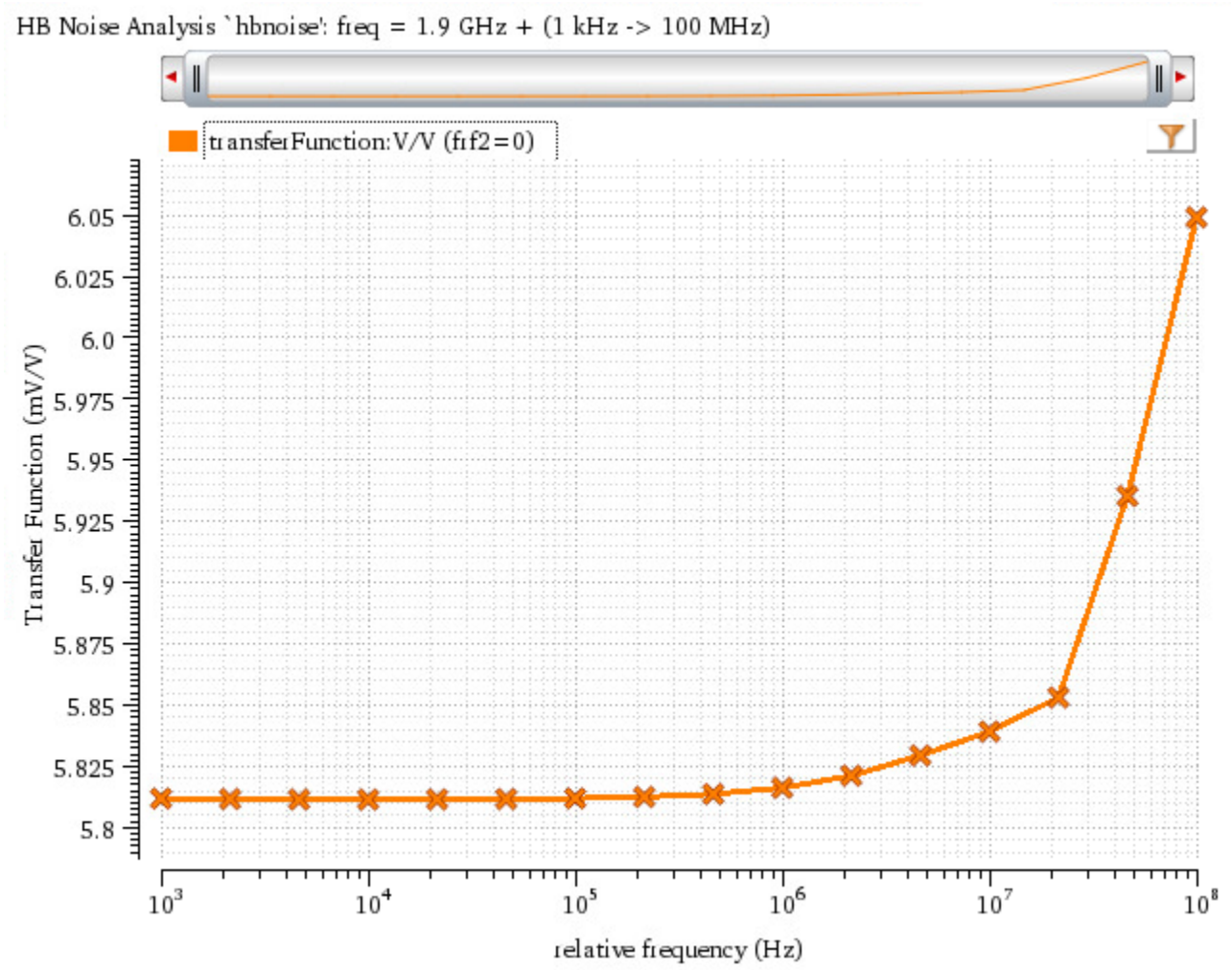
The following example creates a Waveform window and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example plots the waveform `wave1`, which represents the transfer function, in the Waveform window `win1`.

```
awvPlotWaveform(
                win1
                list(wave1)
                ?expr list("transferFunction:V/V")
                ?color list("y6")
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list("x")
                )
```

```
=> t
```



The following example creates a waveform object that represents the plot for transfer function with y-axis unit specified as `dB`.

```
wave2=rfTransferFunction("dB" ?result "hbnoise")
=> srrWave:0x35ff7100
```

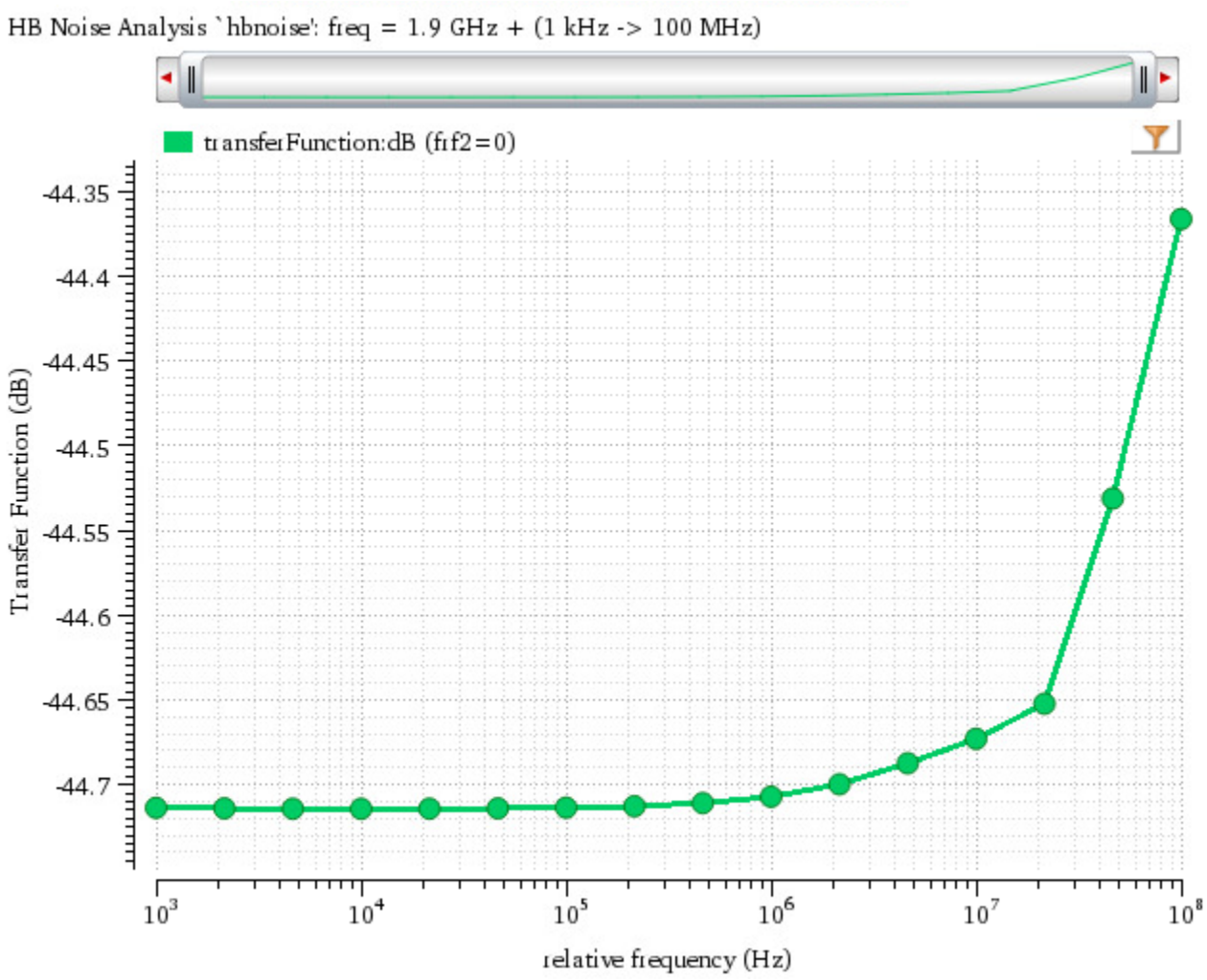
The following example creates another Waveform window and returns its window ID.

```
win2=awvCreatePlotWindow()
=> window:4
```

The following example plots the waveform `wave2`, which represents the transfer function, in the Waveform window `win2`.

```
awvPlotWaveform(
                win2
```

```
                list(wave2)
                ?expr list("transferFunction:dB")
                ?color list("y18")
                ?lineType list("line")
                ?lineStyle list("solid")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list(".")
                )
=> t
```



rfWrlsCcdfValues

```
rfWrlsCcdfValues(
     t_sig
     )
     => o_waveform / f_avgPower / f_peakPower / nil
```

Description

Plots the complementary cumulative distribution function (CCDF) curve or calculates the average or peak power from the results of an envelope (ENVLP) wireless simulation.

Arguments

t_sig	Specifies the value to be calculated or plotted. For example, WPRB2.ccdf, WPRB2.avgPower, or WPRB2.peakPower.

Value Returned

o_waveform	Plots the CCDF curve if the argument *t_sig* is set to WPRB*x*.ccdf.
f_avgPower	Calculates the average power if the argument *t_sig* is set to WPRB*x*.avgPower.
f_peakPower	Calculates the peak power if the argument *t_sig* is set to WPRB*x*.peakPower.
nil	Indicates an error.

Examples

The following example opens simulation results of wireless envelope analysis stored in the specified results directory.

```
openResults("/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/Wireless/psf")
=> "/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/Wireless/psf"
```

The following examples create waveform objects `wave1` and `wave2` representing waveforms of CCDF curves measured by wireless probes `WPRB1` and `WPRB2`, respectively. These wireless probes are inserted into the input and output of the amplifier.

```
wave1=rfWrlsCcdfValues("WPRB1.ccdf")
=> srrWave:0x35caa070
wave2=rfWrlsCcdfValues("WPRB2.ccdf")
=> srrWave:0x35caa0a0
```

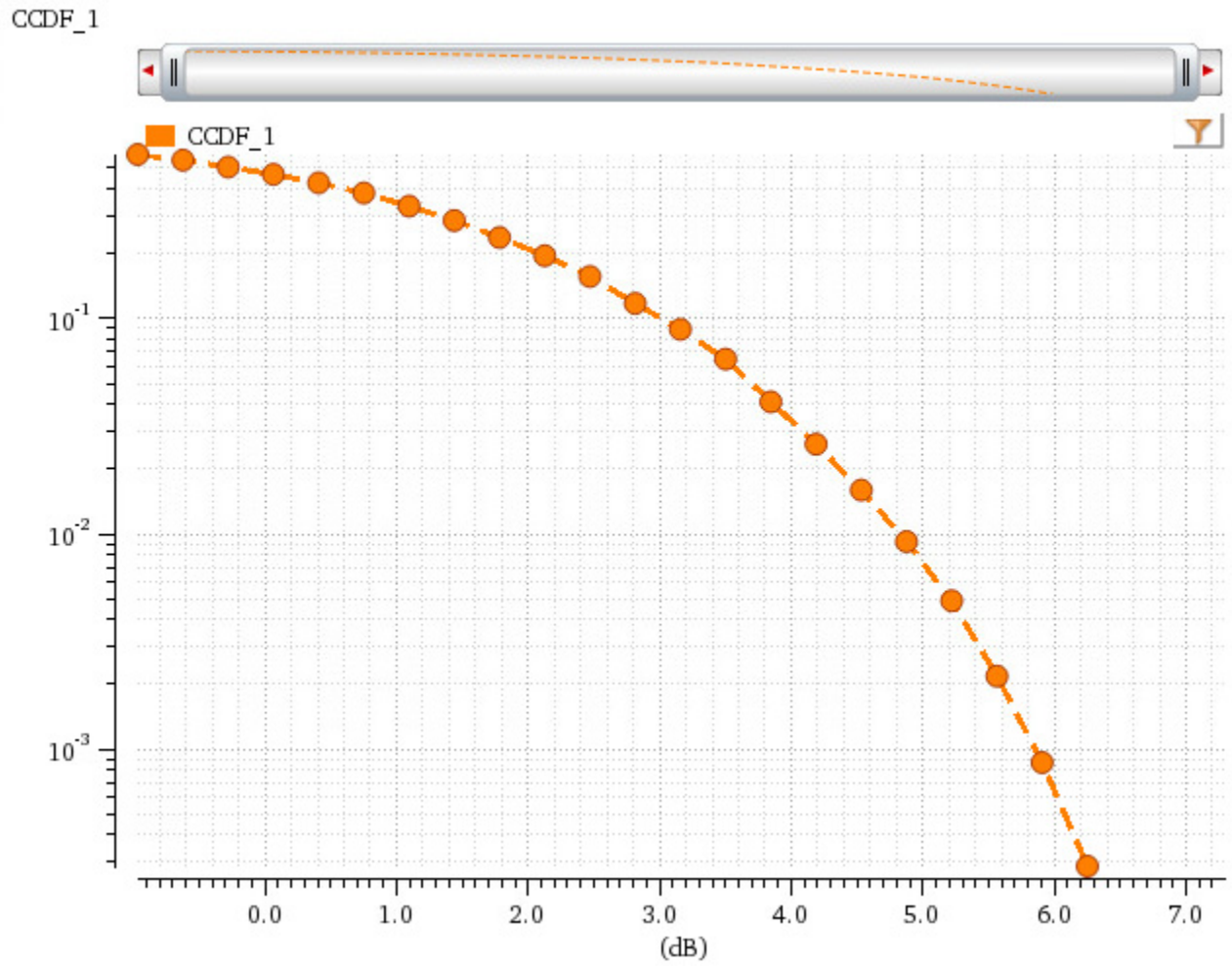
The following example creates a Waveform window and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example plots the CCDF curve represented by the waveform object `wave1` in the Waveform window `win1`.

```
awvPlotWaveform(
                win1
                list(wave1)
                ?expr list("CCDF_1")
                ?color list("y6")
                ?index list(1)
                ?lineType list("line")
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list(".")
                )
```

```
=> t
```



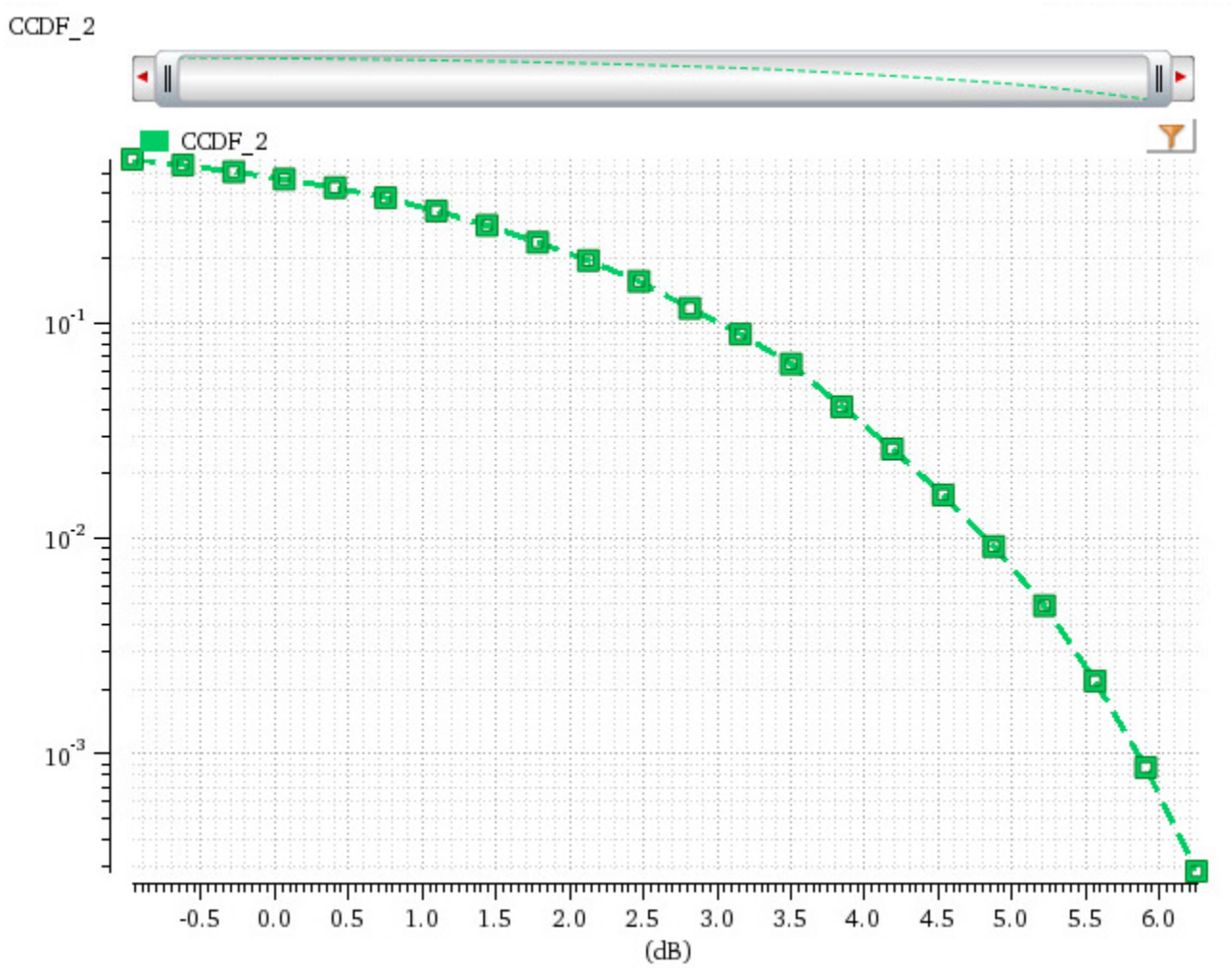
The following example creates a Waveform window and returns its window ID.

```
win2=awvCreatePlotWindow()
=> window:4
```

The following example plots the CCDF curve represented by the waveform object `wave1` in the Waveform window `win1`.

```
awvPlotWaveform(
                win2
                list(wave2)
                ?expr list("CCDF_2")
                ?color list("y18")
                ?index list(1)
                ?lineType list("line")
```

```
                ?lineStyle list("dash")
                ?lineThickness list("thick")
                ?showSymbols list(t)
                ?dataSymbol list(5)
                )
=> t
```



The following examples return the average and peak power measured by `WPRB2.peakPower`, and `WPRB2.avgPower`, respectively.

```
rfWrlsCcdfValues("WPRB2.peakPower")
=> 28.49793
rfWrlsCcdfValues("WPRB2.avgPower")
=> 22.59117
```

rfWrlsCim3Value

```
rfWrlsCim3Value(
     s_probe
     )
     => f_CIM3 / nil
```

Description

Returns the value of third-order counter-intermodulation (CIM3) calculated for the specified probe.

Arguments

s_probe	Name of the probe for which CIM3 needs to be calculated.

Value Returned

f_CIM3	CIM3 value.
nil	CIM3 value cannot be calculated because of an error.

Examples

The following example opens simulation results of wireless envelope analysis stored in the specified results directory.

```
openResults("/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/Wireless/psf")
=> "/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/Wireless/psf"
```

The following examples return values of CIM3 calculated for the specified wireless probes `WPRB1` and `WPRB2`, respectively. These wireless probes are inserted into the input and output of the amplifier.

```
rfWrlsCim3Value("WPRB1")
=> -45.02799
rfWrlsCim3Value("WPRB2")
=> -25.91163
```

rfWrlsCim5Value

```
rfWrlsCim5Value(
     s_probe
     )
     => f_CIM5 / nil
```

Description

Returns the value of fifth-order counter-intermodulation (CIM5) calculated for the specified probe.

Arguments

s_probe	Name of the probe for which CIM5 needs to be calculated.

Value Returned

f_CIM5	CIM5 value.
nil	CIM5 value cannot be calculated because of an error.

Examples

The following example opens simulation results of wireless envelope analysis stored in the specified results directory.

```
openResults("/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/Wireless/psf")
=> "/servers/user/testcase/simulation/lib/cell/maestro/results/maestro/
ExplorerRun.0/1/Wireless/psf"
```

The following examples return values of CIM5 calculated for the specified wireless probes `WPRB1` and `WPRB2`, respectively. These wireless probes are inserted into the input and output of the amplifier.

```
rfWrlsCim5Value("WPRB1")
=> -43.87476
rfWrlsCim5Value("WPRB2")
=> -25.94965
```

rfWrlsMeasContour

```
rfWrlsMeasContour(
     t_sig
     [ ?maxValue n_maxVaue ]
     [ ?minValue n_minValue ]
     [ ?numCont n_numCont ]
     [ ?closeCont g_closeCont ]
     [ ?modifier t_modifier ]
     )
     => n_family / nil
```

Description

Draws contours of system measurements from simulation results of envelope loadpull analysis.

Arguments

t_sig

Name of the system metrics to be analyzed.

For example:

- Adjacent Channel Power Ratio (ACPR)
- Bit Error Rate (BER)
- Error Vector Magnitude (EVM)
- Frequency Shift Keying Error (FSK Error)
- Main Channel Mean Power (MCP)
- Peak-to-Average Power Ratio (PAPR)
- RMS Voltage

?maxValue *n_maxVaue*

The largest value of the contour to be drawn.

If you do not specify the value of this argument, the largest value of the contour from simulation results is used.

?minValue *n_minValue*

The smallest value of the contour to be drawn.

If you do not specify the value of this argument, the smallest value of the contour from simulation results is used.

`?numCont` *`n_numCont`*

Number of contours to be drawn.

`?closeCont` *`g_closeCont`*

Specifies whether to draw a closed contour.

Valid values are:

- `t`: Draws a closed contour.
- `nil`: Draws an open contour.

`?modifier` *`t_modifier`*

Unit modifier.

Depending on the system metric to be analyzed, valid values can be one of the following:

- `dBm/50ohm`
- `Watt/50ohm`
- `dB20`
- `RMS Volt`
- `dBm`
- `Watt`
- `Percent`
- `dBunit`

Value Returned

`n_family`	Family of wave representing the contours drawn for the specified system measurement.
`nil`	Indicates an error.

Examples

The following example opens simulation results of envelope loadpull analysis stored in the specified directory.

```
openResults("/servers/user/ENVLPLoadpull/simulation/lib/cell/maestro/results/
maestro/ExplorerRORun.0/1/lib:cell:1/psf")
```

```
=> "/servers/user/ENVLPLoadpull/simulation/lib/cell/maestro/results/maestro/
ExplorerRORun.0/1/lib:cell:1/psf"
```

The following example lists the results available in the currently open results directory.

```
results()
=> (envlpldp model instance output designParamVals
    primitives subckts variables
)
```

The following example selects the result `envlpldp`.

```
selectResults('envlpldp)
=> stdobj@0x31e55ad0
```

The following example lists the outputs available in the selected result.

```
outputs()
=>
("WPRB1.rmsvol" "WPRB0.rmsvol" "LOAD.rmsvol" "WPRB1.ber.1" "WPRB1.fskerr.1"
    "WPRB0.ber.1" "WPRB0.fskerr.1" "WPRB1.acpr.left1" "WPRB1.acpr.right1"
    "WPRB1.mcp" "WPRB0.acpr.left1" "WPRB0.acpr.right1" "WPRB0.mcp"
    "LOAD.acpr.left1" "LOAD.acpr.right1" "LOAD.mcp" "WPRB1.papr"
    "WPRB0.papr" "LOAD.papr" "/Gamma"
)
```

The following example creates a waveform object `wave1`, which represents `9` constant MCP contours for the measurement `WPRB0.mcp`. In this example, note that the largest and smallest values of the contours are taken from the simulation results of envelope loadpull analysis.

```
wave1=rfWrlsMeasContour("WPRB0.mcp" ?numCont 9 ?closeCont t ?modifier "dBm/50ohm")
=> srrWave:0x365ec040
```

The following example creates a Waveform window and returns its window ID.

```
win1=awvCreatePlotWindow()
=> window:3
```

The following example sets display mode of the Waveform window `win1` to `smith`.

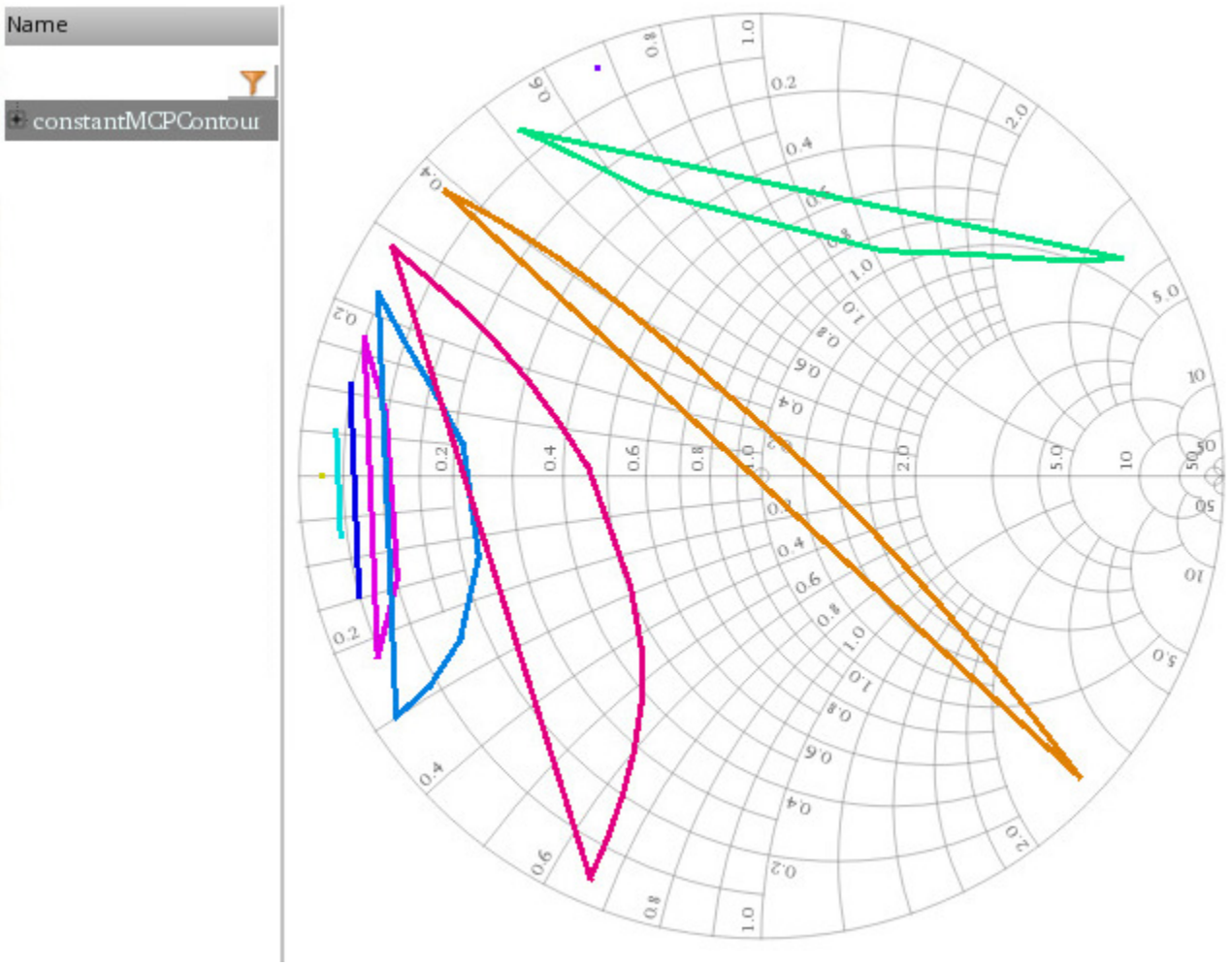
```
awvSetDisplayMode(win1 "smith")
=> t
```

The following example sets the Smith display of the Waveform window `win1` to `impedance`.

```
awvSetSmithModeType(win1 "impedance")
=> t
```

The following example plots the waveform object `wave1` in the Waveform window `win1`.

```
awvPlotWaveform(
                win1
                list(wave1)
                ?expr list("constantMCPContour")
                )
=> t
```



The following example creates a waveform object `wave2`, which represents `9` constant RMS voltage contours for the measurement `WPRB1.rmsvol`. In this example, note that the largest and smallest values of contours are taken from the simulation results of envelope loadpull analysis. The unit modifier is `Watt/50ohm`.

```
wave2=rfWrlsMeasContour("WPRB1.rmsvol" ?numCont 9 ?closeCont t ?modifier "Watt/
50ohm")
=> srrWave:0x365ec170
```

The following example creates a Waveform window and returns its window ID.

```
win2=awvCreatePlotWindow()
window:4
```

The following example sets display mode of the Waveform window `win2` to `smith`.

```
awvSetDisplayMode(win2 "smith")
=> t
```

The following example sets the Smith display of the Waveform window `win2` to `impedance`.

```
awvSetSmithModeType(win2 "impedance")
=> t
```

The following example plots the waveform object `wave2` in the Waveform window `win2`.

```
awvPlotWaveform(
                win2
                list(wave2)
                ?expr list("constRMSVoltageContour")
                )
```

```
=> t
```