

SIMetrix

SPICE and Mixed Mode Simulation

Script Reference Manual

Copyright ©1992-2009 SIMetrix Technologies Ltd.

Contact

SIMetrix Technologies Ltd., Terence House, 24 London Road,
Thatcham, RG18 4LQ, United Kingdom

Tel: +44 1635 866395
Fax: +44 1635 868322
Email: support@simetrix.co.uk
Internet <http://www.simetrix.co.uk>



Copyright © SIMetrix Technologies Ltd. 1992-2009
SIMetrix Script Reference Manual 3/4/09

Chapter 1 Introduction

Chapter 2 The SIMetrix Script Language

A Tutorial 20

 Example 1: Hello World! 20

 Example 2: An Introduction to Loops 20

 Example 3: Cross probing 22

 Example 4: Making a Parts List 23

Variables, Constants and Types 25

 Variable names 25

 Types 26

 Constants 26

 Creating and Assigning Variables 27

 New line Character 27

 Vectors 27

 Scope of Variables. Global Variables 28

 Empty Values 28

 Empty Strings 29

 Quotes: Single and Double 29

Expressions 30

 Operators 30

 Functions 31

 Braced Substitutions 32

 Bracketed Lists 32

 Type Conversion 33

 Aliases 33

Statements and Commands 33

 Commands 34

 Command Switches 34

 If Statement 34

 While Statement 35

 For Statement 36

 Script Statement 36

 Exit Statement 37

Accessing Simulation Data 37

 Overview 37

 Groups 37

 Multi-division Vectors 38

 Vector References 42

 Physical Type 42

User Interface to Scripts 42

 Dialog Boxes 42

 User Control of Execution 43

Errors.....	43
Syntax Errors	44
Execution Errors.....	44
Error Messages.....	44
Executing Scripts.....	44
Script Arguments.....	45
Built-in Scripts	46
Debugging Scripts	46
Startup Script	47

Chapter 3 Function Summary

Function Summary	48
Functions by Application	67
Unsupported Functions	72

Chapter 4 Function Reference

abs	73
ACSourceDialog	73
AddConfigCollection	74
AddGraphCrossHair	74
AddModelFiles	75
AddPropertyDialog	75
AddRemoveDialog	76
AddSymbolFiles	77
arg	77
arg_rad	78
Ascii	78
AssociateModel	78
atan	79
atan_deg	79
BoolSelect	79
Branch.....	80
CanOpenFile	81
ChangeDir	81
Char	82
ChooseDir	82
ChooseDirectory.....	83
Chr.....	83
CloseEchoFile	83
CloseFile	83
CloseSchematic	84
CollateVectors.....	84

CompareSymbols	85
ComposeDigital	86
ConvertLocalToUnix	88
ConvertUnixToLocal	88
CopyURL	89
cos	90
cos_deg	90
CreateDiodeDialog	91
CreateLockFile.....	91
CreateShortcut.....	92
CreateTimer	93
CyclePeriod	93
cv	94
Date	94
db	95
DCSourceDialog	95
DefineADCDialog	95
DefineArbSourceDialog	96
DefineBusPlotDialog.....	97
DefineCounterDialog	98
DefineCurveDialog	99
DefineDACDialog	101
DefineFourierDialog	102
DefineIdealTxDialog	103
DefineLaplaceDialog	104
DefineLogicGateDialog	105
DefinePerfAnalysisDialog	106
DefineRegisterDialog	106
DefineRipperDialog	107
DefineSaturableTxDialog	108
DefineSimplisMultiStepDialog	109
DefineShiftRegDialog	109
DeleteConfigCollection	110
DeleteTimer	110
DescendDirectories	111
DescendHierarchy	111
diff	112
EditArcDialog	112
EditAxisDialog	113
EditBodePlotProbeDialog	114
EditCrosshairDimensionDialog	115
EditCurveMarkerDialog	116
EditDeviceDialog	117
EditDigInItDialog	119

EditFreeTextDialog	119
EditGraphTextBoxDialog	119
EditLegendBoxDialog	120
EditObjectPropertiesDialog	120
EditPinDialog	122
EditPotDialog	122
EditProbeDialog	123
EditPropertyDialog	125
EditReactiveDialog	126
EditSelect	128
EditTimer	128
EditWaveformDialog	129
EnterTextDialog	131
EscapeString	131
ev	132
Execute	133
ExistDir	133
ExistFunction	133
ExistSymbol	134
ExistVec	134
exp	135
fft	135
Field	135
FilterFile	136
FindModel	136
FIR	137
Floor	138
Floorv	138
FormatNumber	138
Fourier	139
FourierOptionsDialog	140
FourierWindow	140
FullPath	141
GenPrintDialog	141
GetActiveWindow	143
GetAllCurves	143
GetAllSymbolPropertyNames	143
GetAllYAxes	144
GetAnalysisInfo	144
GetAnalysisLines	145
GetAxisCurves	146
GetAxisLimits	146
GetAxisType	146
GetAxisUnits	147

GetChildModulePorts.....	147
GetColours.....	148
GetColourSpec	148
GetConfigLoc.....	148
GetConnectedPins	149
GetConvergenceInfo.....	150
GetCurDir.....	151
GetCurrentGraph	151
GetCursorCurve.....	151
GetCurveAxis	152
GetCurveName	152
GetCurves.....	152
GetCurveVector	153
GetDatumCurve.....	153
GetDeviceDefinition	154
GetDeviceInfo	155
GetDeviceParameterNames	156
GetDotParamNames	157
GetDotParamValue.....	157
GetDriveType	158
GetEmbeddedFileName	159
GetEnvVar	159
GetEthernetAddresses	159
GetF11Lines	160
GetFile	161
GetFileCd.....	161
GetFileExtensions	161
GetFileInfo	162
GetFileSave.....	163
GetFonts.....	163
GetFontSpec	163
GetFreeDiskSpace	164
GetGraphObjects	164
GetGraphObjPropNames	165
GetGraphObjPropValue	165
GetGraphTabs	166
GetGraphTitle	166
GetGroupInfo	166
GetGroupStepParameter	167
GetGroupStepVals.....	167
GetInstanceBounds	168
GetInstanceParamValues.....	169
GetInstancePinLocs	170
GetInstsAtPoint	170

GetInternalDeviceName	171
GetKeyDefs	171
GetLastCommand	172
GetLastError	172
GetLegendProperties	173
GetLibraryModels	173
GetLicenseInfo	174
GetLicenseStats	175
GetLine	176
GetLongPathName	176
GetMenuItems	177
GetModelFiles	177
GetModelName	178
GetModelPropertyNames	178
GetModelPropertyValues	179
GetModelType	179
GetModifiedStatus	180
GetNamedSymbolPins	180
GetNamedSymbolPropNames	181
GetNamedSymbolPropValue	181
GetNearestNet	182
GetNonDefaultOptions	182
GetNumCurves	183
GetOpenSchematics	183
GetOption	183
GetPath	184
GetPlatformFeatures	184
GetPrinterInfo	185
GetPrintValues	186
GetReadOnlyStatus	186
GetSchematicVersion	186
GetSchemTitle	187
GetSelectedCurves	187
GetSelectedGraphAnno	187
GetSelectedYAxis	188
GetShortPathName	188
GetSimConfigLoc	188
GetSimetrixFile	189
GetSimplisAbortStatus	190
GetSimplisExitCode	190
GetSimulationErrors	190
GetSimulationInfo	191
GetSimulationSeeds	191
GetSimulatorEvents	191

GetSimulatorMode	193
GetSimulatorOption	194
GetSimulatorStats	194
GetSimulatorStatus	195
GetSoaDefinitions	196
GetSoaMaxMinResults	196
GetSoaOverloadResults	197
GetSoaResults	197
GetSymbolArcInfo	198
GetSymbolFiles	198
GetSymbolInfo	198
GetSymbolOrigin	199
GetSymbolPropertyInfo	200
GetSymbolPropertyNames	200
GetSymbols	201
GetSymbolText	202
GetTimerInfo	203
GetSystemInfo	203
GetToolButtons	204
GetUncPath	205
GetUserFile	205
GetVecStepParameter	207
GetVecStepVals	208
GetWindowNames	208
GetXAxis	209
GraphLimits	209
GroupDelay	210
Groups	210
Hash	210
HasLogSpacing	211
HasProperty	211
HaveFeature	212
HaveInternalClipboardData	212
HierarchyHighlighting	213
HighlightedNets	213
Histogram	213
Iff	214
IIR	215
im	216
imag	217
InputGraph	217
InputSchem	217
Instances	217
InstNets	218

InstNets2	219
InstPins	219
InstPoints	220
InstProps	221
Integ	222
Interp	223
IsComplex	224
IsComponent	224
IsFullPath	224
IsModelFile	225
IsNum	225
IsOptionMigrateable	225
IsSameFile	226
IsScript	226
IsStr	227
JoinStringArray	227
Length	228
ListDirectory	228
In	229
Locate	229
log	229
log10	230
mag	230
magnitude	230
MakeDir	230
MakeLogicalPath	231
MakeString	231
ManageMeasureDialog	232
Max	232
Maxidx	233
Maxima	233
Maximum	234
mean	234
Mean1	235
MeasureDialog	235
MessageBox	236
Mid	237
Minidx	238
Min	238
Minima	238
Minimum	239
ModelLibsChanged	239
Navigate	240
NearestInst	240

NetName	241
NetNames	241
NetWires	242
NewPassiveDialog	242
NewValueDialog	244
norm	245
NumDivisions	246
NumElems	246
OpenEchoFile	246
OpenFile	247
OpenSchem	247
OpenSchematic	249
Parse	250
ParseAnalysis	250
ParseParameterString	251
ParseSimplisnit	253
PathEqual	253
ph	254
phase	254
phase_rad	254
PhysType	254
PinName	255
Progress	256
Probe	257
ProcessingAccelerator	257
PropFlags	257
PropFlags2	258
PropValue	259
PropValues	260
PropValues2	261
PutEnvVar	262
PWLDialog	262
QueryData	264
RadioSelect	265
Range	266
re	266
ReadClipboard	267
ReadConfigCollection	267
ReadConfigSetting	267
ReadFile	268
ReadIniKey	269
ReadRegSetting	270
ReadSchemProp	270
RemoveSymbolFiles	271

real	272
Ref	272
RefName	272
RelativePath	272
RemoveConfigCollection.....	273
RemoveModelFile	273
ResolveGraphTemplate	274
ResolveTemplate	275
RestartTranDialog	275
Rms	276
RMS1	276
rnd	276
RootSumOfSquares	277
SaveSpecialDialog	277
Scan	278
ScriptName.....	278
Search	279
SearchModels	279
Seconds	280
SelectAnalysis.....	280
SelectColourDialog	281
SelectColumns	281
SelectCount	282
SelectDevice	282
SelectDialog	283
Select2Dialog	284
SelectedProperties.....	285
SelectedWires	285
SelectFontDialog	286
SelectRows	286
SelectSimplisAnalysis	287
SelGraph	287
SelSchem	287
SelectSymbolDialog	288
SetReadOnlyStatus.....	289
Shell	290
ShellExecute	292
sign	293
SimatrixFileInfo.....	293
SimulationHasErrors	294
sin	294
sin_deg	295
Sleep	295
Sort	295

SortIdx	296
SourceDialog	296
SplitPath	297
SprintfNumber.....	297
sqrt	298
Str	298
StringLength	298
StrStr	299
SubstChar.....	299
SubstString	300
SumNoise	300
SymbolGen	301
SymbolInfoDialog	302
SymbolLibraryManagerDialog	303
SymbolName	303
SymbolNames	304
SymbolPinOrder	305
SystemValue.....	305
TableDialog.....	305
tan	307
tan_deg	307
TemplateGetPropValue	307
TemplateResolve.....	308
Time	308
TransformerDialog	308
TranslateLogicalPath.....	310
TreeListDialog	310
TRUE	312
Truncate	312
Units	313
unitvec	313
UpDownDialog	314
UserParametersDialog	314
Val	315
ValueDialog	316
Vec	317
vector	317
VectorsInGroup	318
VersionInfo.....	318
WirePoints	319
Wires	320
WriteConfigSetting	321
WriteF11Lines.....	321
WriteIniKey	322

WriteRawData	323
WriteRegSetting	323
WriteSchemProp	324
XCursor	324
XDatum	325
XFromY	325
XY	325
YCursor	326
YDatum	326
YFromX	326

Chapter 5 Command Summary

Notation	327
Command Summary	328
Commands by Application.....	335

Chapter 6 Command Reference

Abort.....	338
AbortSIMPLIS.....	338
About	338
AddArc.....	338
AddCirc.....	339
AddCurveMarker	339
AddFreeText.....	340
AddGraphDimension	340
AddLegend	341
AddLegendProp	342
AddPin.....	343
AddProp	344
AddSeg.....	346
AddSymbolProperty	346
AddTextBox.....	347
Anno	347
AppendTextWindow	348
Arguments	348
BuildDefaultOptions.....	348
Cancel	349
Cd.....	349
ChangeArcAttributes	349
ChangeSymbolProperty	349
ClearMessageWindow	350
Close	350

CloseGraphSheet	350
ClosePrinter	350
CloseSheet	350
CloseSimplisStatusBox.....	351
CollectGarbage	351
CompareSymbolLibs	351
Copy	351
CopyClipGraph	352
CopyClipSchem	352
CopyFile.....	353
CopyLocalSymbol.....	353
CreateFont	354
CreateGroup	354
CreateSym	354
CreateToolBar	355
CreateToolButton.....	356
CursorMode	357
Curve	358
CurveEditCopy.....	361
DefButton	361
DefineToolBar	362
DefKey	365
DefMenu	368
Del	372
DelCrv	372
Delete	372
DeleteAxis.....	372
DeleteGraphAnno	373
DeleteSymbolProperty	373
DelGroup	373
DelLegendProp	373
DelMenu	374
DelProp	374
DelSym	374
DelSymLib	375
Detach	375
Discard.....	375
Display	375
DrawArc	376
DrawPin	376
Echo.....	377
EditColour	377
EditCopy	377
EditCut.....	378

EditFile	378
EditFont	378
EditPaste	378
EditPin	378
EndSym	379
ExecuteMenu	379
Execute	379
Focus	380
FocusShell	380
GraphZoomMode	381
Help	381
HideCurve	381
HighlightCurve	381
Hint	382
HourGlass	382
ImportSymbol	383
Inst	383
KeepGroup	384
Let	385
Listing	385
ListModels	385
ListOptions	385
ListStdButtonDefs	386
ListStdKeys	386
LoadModelIndex	386
MakeAlias	386
MakeCatalog	387
MakeSymbolScript	387
MakeTree	388
MCD	388
MD	388
Message	388
MessageBox	388
Move	388
MoveCurve	389
MoveFile	389
MoveMenu	389
MoveProperty	389
Netlist	390
NewAxis	392
NewGraphWindow	392
NewGrid	392
NewPrinterPage	392
NewSchem	392

NewSymbol.....	393
NoPaint.....	393
NoUndo.....	393
OpenGraph.....	393
OpenGroup.....	394
OpenPrinter.....	395
OpenRawFile.....	395
OpenSchem.....	396
OpenSimplisStatusBox.....	396
OptionsDialog.....	397
Pan.....	397
Pause.....	397
PinDef.....	397
PlaceCursor.....	398
Plot.....	398
PreProcessNetlist.....	400
PrintGraph.....	401
PrintSchematic.....	401
Probe.....	402
Prop.....	402
Protect.....	405
Quit.....	405
RD.....	405
ReadLogicCompatibility.....	406
RebuildSymbols.....	407
Redirect.....	407
RedirectMessages.....	407
Redo.....	408
RegisterUserFunction.....	408
RenameLibs.....	408
RepeatLastMenu.....	409
Reset.....	409
RestartTran.....	409
RestDesk.....	410
Resume.....	410
RotInst.....	410
Run.....	410
RunSIMPLIS.....	413
Save.....	413
SaveAs.....	414
SaveDesk.....	414
SaveGraph.....	415
SaveGroup.....	415
SaveRhs.....	415

SaveSnapshot	416
SaveSymbol	416
SaveSymLib	417
ScriptAbort.....	417
ScriptPause	418
ScriptResume.....	418
ScriptStep.....	418
Select	419
SelectCurve	419
SelectGraph	420
SelectLegends.....	420
SelectSimulator	420
Set.....	420
SetCurveName.....	421
SetGraphAnnoProperty	421
SetGroup	421
SetHighlight	421
SetOrigin	422
SetReadOnly	422
SetRef	423
SetSnapGrid.....	423
SetSymbolOriginVisibility	423
SetToolBarVisibility	423
SetUnits.....	424
SetWireColour	424
Shell	425
ShellOld.....	425
Show	426
ShowCurve.....	427
ShowSimulatorWindow	427
SizeGraph	427
Stats	428
TemplateEditProperty.....	428
TemplateSetValue.....	428
TextWin	429
Title.....	429
Trace	429
Undo.....	429
UndoGraphZoom.....	430
UnHighlightCurves	430
UnLet.....	430
Unprotect.....	430
Unselect	430
UnSet	430

UpdateAllSymbols	431
UpdateProperties	431
UpdateSymbol	431
ViewFile	431
Wait.....	432
Where	432
Wire	432
WireMode	432
WriteImportedModels.....	432
Zoom.....	432

Chapter 7 Applications

User Interface	434
User Defined Key and Menu Definitions.....	434
Rearranging or Renaming the Standard Menus.....	434
Menu Shortcuts	434
Editing Schematic Component Values	435
Modifying Internal Scripts	435
Custom Curve Analysis	435
Adding New Functions.....	435
‘measure’, ‘measure_span’ Scripts.....	436
An Example: The ‘Mean’ Function.....	436
Automating Simulations	437
Overview.....	437
Running the Simulator.....	437
Changing Component Values or Test Conditions ..	437
Organising Data Output from Automated Runs.....	439
An Advanced Example - Reading Values from a File	439
Schematic Symbol Script Definition.....	442
Defining New Symbol	443
Symbol Definition Format	443
How Symbols are Stored.....	445
Data Import and Export.....	445
Importing Data	446
Exporting Data.....	446
Launching Other Applications.....	446
Data Files Text Format	447
Graph Objects.....	448
Overview.....	448
Object Types	448
Properties	448
Graph Object Identifiers - the “ID”	449
Symbolic Values	450

Objects and Their Properties	450
Graph Co-ordinate Systems.....	460
Event Scripts	460
on_graph_anno_doubleclick	461
on_accept_file_drop.....	461
on_schem_double_click.....	461
User Defined Script Based Functions	461
Overview	461
Defining the Function	462
Registering the Script.....	462
Example	462
User Defined Binary Functions.....	462
Overview	462
Documentation	463
Non-interactive and Customised Printing	463
Overview	463
Procedure.....	463
Example	463
Schematic Template Scripts.....	464
Overview	464
Defining a Symbol for a Template Script.....	465
When is the Template Script Called?	465
The Template Script.....	465
Template Commands and Functions	465
Creating and Modifying Toolbars	466
Modifying Existing Toolbars and Buttons	466
Redefining Button Commands	467
Defining New Buttons and Editing Buttons	467
Creating New Toolbars	469
Pre-defined Buttons	469
Custom Dialog Boxes.....	472
Overview	472
Starting "SIMetrix Dialog Designer"	472
Developing Dialogs	472
The Widgets.....	473
Using Geometry Management	476
Examples	477
ExecuteDialog Function	478
Performance.....	478

Chapter 1 Introduction

SIMetrix features a simple interpreted script language, loosely based on BASIC, in which most of the user interface is written.

This manual provides the means for users sympathetic to the concept of computer programming to develop their own scripts or to adapt the user interface by modifying the internal scripts.

We have identified three main applications for script development although there may be others we haven't thought of. These are:

1. User interface modification perhaps to suit individual taste or for specialised applications.
2. Automated simulations. For example, you may have a large circuit which for which you need to run a number of tests. The simulations take along time so you would like to run them overnight or over a weekend. A simple script can perform this task.
3. Specialised analysis. The curve analysis functions supplied with SIMetrix are all implemented using scripts. You can write your own to implement specialised functionality. Also the goal functions used for performance and histogram analysis are "user defined functions" and are actually implemented as scripts. More goal functions may be added for special applications.

The scripting language is supported by about 420 functions and 230 commands that provide the interface to the SIMetrix core as well as some general purpose functionality.

As well as the built-in functions, a tool kit is available that allows you to develop your own functions in 'C' or 'C++'.

Chapter 2 The SIMetrix Script Language

A Tutorial

Example 1: Hello World!

Any one who has learnt the 'C' programming language will be familiar with the now celebrated "Hello World" program - possibly the simplest program that can be written. Here we will write and execute a SIMetrix "Hello World" script.

The script is simple:

```
echo "Hello World!"
```

To execute and run this script start by selecting the menu File|Scripts|New Script this simply launches a text editor with the script directory as its working directory. (On Linux you will probably need to set up a text editor. The default is gedit - if this is unsuitable use File|Options|General... then select File Locations tab and enter a suitable text editor.) Type:

```
echo "Hello World!"
```

Now save the text to a file called hello.sxscr.

To execute the script, type "hello" at the command line. You should see the message:

```
Hello World!
```

Appear in the message window. A script is executed by typing its filename at the command line. Note that the filename is case sensitive under Linux. If the file has the extension .sxscr the extension may be omitted. You can also assign a key or menu to execute a script. Type at the command line:

```
DefKey F6 HELLO
```

Now press the F6 key. The message should appear again. For information on defining menus see ["User Defined Key and Menu Definitions" on page 434](#)

Example 2: An Introduction to Loops

This example adds up all the elements in a vector (or array). To create a vector we will run a simulation on one of the example circuits. The whole process will be put into a script except opening the schematic which we will do manually. (But this can be done from a script as well).

To start with, open the example circuit General/AMP.sxsch. Make sure it is selected to run a transient analysis.

Now select File|Scripts|New Script. This will open a text editor with the current directory set to the SCRIPT. Type in the following:

```
Anno
Netlist design.net
Run design.net

let sum = 0
for idx=0 to length(vout)-1
    let sum = sum + vout[idx]
next idx

echo The sum of all values in vout is {sum}
```

Save the script to the file name SUM.sxsct. Now type SUM at the command line. A simulation will run and the message:

```
The sum of all values in vout is -6.1663737561
```

Should appear in the message window. The exact value given may be different if you have modified the circuit or set up different model libraries.

This script introduces four new concepts:

1. For loops
2. Braced substitutions ({sum} in the last line)
3. Vectors (or arrays)
4. Accessing simulation data

Let's go through this script line by line.

The first three lines carry out the simulation and in fact something similar is done each time a simulation is run using the menu or F9 key. Anno annotates the netlist to ensure that there are no duplicate component references. Netlist design.net generates a netlist of the circuit and saves it in a file called design.net. Finally Run design.net runs the simulation on the netlist design.net.

The line

```
let sum = 0
```

creates and initialises the variable sum which will ultimately hold the final result. The next three lines is a simple *for statement*. The variable idx is incremented by one each time around the loop starting at zero and ending at length(vout)-1. vout is a variable - actually a vector - which was generated by the simulator and holds the simulated values of the voltage on the VOUT net. This net is marked with a terminal symbol. length(vout) returns the number of elements in vout. (1 is subtracted because idx starts at 0). In the line:

```
let sum = sum + vout[idx]
```

vout[idx] is an indexed expression which returns element number idx of the vector vout. sum is of course the accumulative total. The final line:

```
echo The sum of all values in vout is {sum}
```

contains the *braced substitution* {sum}. sum is evaluated and the result replaces expression and the braces. See “[Braced Substitutions](#)” on page 32 for more information.

Example 3: Cross probing

The standard plotting menus, plot one curve at a time. Here a script is described which repeatedly plots cross-probed curves until the right mouse key is clicked.

```
let start=1
do while probe()
  if start then
    plot {netname()}
  else
    curve {netname()}
  endif
  let start=0
  probe
loop
```

This script introduces *if statements*, *while statements*, *functions* and the features that allow voltage cross-probing, namely the functions [NetName \(page 241\)](#) and [Probe \(page 257\)](#) and the command [Probe \(page 402\)](#).

The script repeatedly executes the statements between `do while` and `loop` until the `probe()` function returns 0 (=FALSE). The `Probe` function changes the cursor shape to an oscilloscope probe but doesn't return until the user presses the left or right mouse key. If the user presses the left key the function returns 1 (=TRUE) and execution continues to the statements inside the loop. If the user presses the right key, the `Probe` function returns 0 (=FALSE) and the loop is completed and the script terminates. In the next 5 lines:

```
if start then
  plot {netname()}
else
  curve {netname()}
endif
```

the first time around the loop `start` is equal to 1 and the `Plot` command is executed. This creates a new graph. Subsequently, `start` is set to zero and the `Curve` command is executed which adds new curves to the graph already created.

The argument to the `Plot` and `Curve` commands, `{netname() }` is a *braced substitution* which we saw in the previous example. The `NetName` function returns a string which is the name of the nearest net to the cursor at the time the function is executed. The function is executed soon after the user presses the left mouse key so the string returned by `NetName` will be the net the user is pointing to. The value returned by `NetName` is a string, but the `Plot` command requires a numeric expression. By putting `netname()` in braces the result of evaluating it is substituted as if it were typed in. So if the user pointed at a net named `VOUT`, `netname()` would return `'VOUT'` and that would be placed after `Plot` or `Curve` i.e. `plot vout` would be executed.

The final command

```
probe
```

calls the `Probe` *command*. This does the same as the `Probe` *function* but doesn't return a result. It is needed because both the `Probe` function and the `Probe` command return on both up and down clicks of the mouse. The second occurrence of `Probe` simply waits for the up click of the mouse button

There are four other functions which are used for cross-probing. These are [GetNearestNet \(page 182\)](#), [NearestInst \(page 240\)](#), [PinName \(page 255\)](#) and [Branch \(page 80\)](#).

Just one final note. `plot {netname() }` won't work for vectors whose name contains certain characters such as arithmetic characters e.g. '+' and '-'. These characters get interpreted as their literal meaning and an error usually results. To plot vectors whose names contain these characters, you should use the `Vec()` function and supply the vector name as a string. E.g.

```
plot Vec(netname())
```

Note that there are no curly braces used here. This is because the `Vec()` function returns a numeric vector containing the actual data to be plotted. The `netname` function returns the *name* of the vector not its actual data.

Example 4: Making a Parts List

This script example displays a list of components in the currently selected schematic with their references and values in the message window.

```
* mk_bom.txt Display parts list in message window
if NOT SelSchem() then
    echo There are no schematics open
    exit all
endif

let refs = PropValues('ref', 'ref')

for idx=0 to length(refs)-1

    let val = PropValues('value', 'ref', refs[idx])

    * check for duplicate ref
    if length(val)==1 then
        echo {refs[idx]} {val}
    else
        echo Duplicate reference {refs[idx]}. Ignoring
    endif
next idx
```

The first line:

```
* do_bom.txt Display parts list in message window
```

is a comment. Any line beginning with a '*' will be ignored.

The next line:

```
if NOT SelSchem() then
```

is the start of an *if statement*. `SelSchem()` is a function which returns 1 if there are schematics open and 0 if there are not. `if NOT SelSchem() then` means 'if there are no schematics open'. This is an initial check that the user has actually opened a schematic.

Script Reference Manual

If there are no schematic open the lines:

```
echo There are no schematics open
exit all
```

will be executed. The first line calls the `echo` command. This echoes to the message window all subsequent text on the same line. The second line is an *exit statement*. In this case it causes execution to abort and the rest of the script will be ignored.

The next line

```
endif
```

terminates the *if statement*. For every `if` there must be a matching `endif` or `end if`.

Normally, of course, we hope the user has opened a schematic and the remainder of the script will be executed. The next line

```
let refs = PropValues('ref', 'ref')
```

calls the `let` command. This expects an assignment expression which it evaluates. In this case it assigns `refs` with the result of the a call to the function [PropValues \(page 260\)](#). In this example it returns the component reference for all instances (i.e. symbols) on the schematic that have one.

The next line ...

```
for idx=0 to length(refs)-1
```

starts a *for loop*. The block of statements between this line and the matching `next` will be repeated with values of `idx` incrementing by 1 each time around the loop until `idx` reaches `length(refs)-1`. The `length` function returns the number of elements in the `refs` variable so the loop is repeated for all elements in `refs`.

The next line is

```
let val = PropValues('value', 'ref', refs[idx])
```

This calls the `PropValues` function again. This time it returns the value of the *value* property for any instance with the property *ref* which has the value `refs[idx]`. Assuming the schematic has been annotated (unique references assigned to all components) the result of this call should be a single value which is assigned to `val`.

The next 2 lines

```
if length(val)==1 then
  echo {refs[idx]} {val}
```

The `if` statement checks that `val` has length one which means that the reference is unique. If it is then the `Echo` command is called which displays on the message window all the text following it. In this instance the `echo` command is followed by two *braced substitutions*. A braced substitution is an expression enclosed in curly braces '{' and '}'. The braces and the enclosed expression are replaced by the result of evaluating the expression as if it had been typed in. Braced substitutions are a very important

feature of the SIMetrix scripting language. Here the result is the component's reference and value are displayed in the message window.

The last part of the for loop is:

```
else
    echo Duplicate reference {refs[idx]}. Ignoring
endif
```

This is executed if the if expression `length(val)==1` is false. This means that there is more than one component with that component reference. A message is output saying that it is being ignored.

The final line

```
next idx
```

terminates the for loop.

Variables, Constants and Types

SIMetrix scripts, like all computer programs, process data stored in variables.

Variables may hold real, complex or string data and may be scalar - possessing only a single value - or single dimension arrays called vectors.

Variable names

Variables names must be a sequence of characters but the first must be non-numeric.

Any character may be used except: `\ " & + - * / ^ < > ' @ { } () [] ! % ; : | =` and spaces.

Although it is legal the following names should be avoided as they are statement keywords:

```
all
do
else
elseif
end
endif
endwhile
exit
for
if
loop
next
script
step
then
to
while
```

Types

Variables may have real, complex or string type. Real and complex are self-explanatory. Strings are a sequence of ASCII characters of any length.

SIMetrix does not have an integer type. Although all numbers are represented internally as floating point values, the format used permits integers to be represented exactly up to values of about 2^{52} .

Constants

These can be real complex or string. Real numbers are represented in the usual way but may also contain the engineering suffixes:

a	10^{-18}
f	10^{-15}
p	10^{-12}
n	10^{-9}
u	10^{-6}
m	10^{-3}
k	10^{+3}
Meg	10^{+6}
G	10^{+9}
T	10^{+12}

Note that engineering suffixes *are not case sensitive*. A common mistake is to use 'M' when what was meant was 'Meg'. 'M' is the same as 'm'.

Complex numbers are represented in the form:

(real, imaginary)

Strings are a sequence of text characters enclosed in single quotation marks. Single quotation marks themselves are represented by two in succession.

Examples

Real:

```
2.3
4.6899
45
1e-3
1.2u
```

Complex

```
(1,1) means 1+i
(2.34,10) means 2.34+10i
```

String

```
'this is a string'
'This is a 'string' '
```

Creating and Assigning Variables

Variables are created and assigned using the `Let` command. For example:

```
Let x=3
```

assigns the value 3 to the variable `x`. Note that `Let` is not optional as it is in most forms of Basic.

You can also assign complex numbers and strings e.g.

```
Let x=(5,1)
Let s='This is a string'
```

All of the above are *scalar* that is they contain only one value. Variables may also be single dimension arrays called *vectors*. Vectors are described below.

New line Character

To enter a new line character use `\\`. If you need a literal double backslash enclose it in quotation marks i.e. `“\\”`. Note however that the use of `\\` doesn't work inside braced substitutions. To use a line feed in a braced substitution, assign the whole string to a variable then put the variable inside the braces. E.g

```
Let error = 'Error:\\Too many nodes selected'
MessageBox {error}
```

Vectors

Vectors can be created using a *bracketed list*, with a function that returns a vector or by the simulator which creates a number of vectors to represent node voltages and device currents. A bracketed list is of the form:

```
[ expression1, expression2, ...]
```

E.g.

```
let v = [1, 3, 9]
```

These are described in more detail on [page 32](#). Functions and simulator vectors are described in following sections.

Vectors, like other variables may also contain strings or complex numbers but all the elements must be the same *type*.

Individual elements of vectors may be accessed using square brackets: `[` and `]`. E.g.

```
let v = [1, 3, 9]
let a = v[2]
```

`a` is assigned 9 in the above example. Index values start at 0 so the first element (1) is `v[0]`.

It is also possible to assign values to individual elements e.g.

```
let v[2] = 5
```

In which case the value assigned must have the same type (i.e. real, complex or string) as the other elements in the vector.

Vectors, like other variables may also contain strings or complex numbers but all the elements must be the same type.

Scope of Variables. Global Variables

Variables created using the `Let` command are only available within the script where the `Let` command was executed. The variable is destroyed when the script is completed and it is not accessible to scripts that the script calls. If, however, the `Let` command was called from the command line, the variable is then *global* and is available to all scripts until it is explicitly deleted with the `UnLet` command.

If a global variable needs to be created within a script, the variable name must be preceded by `global:`. For example:

```
Let global:result = 10
```

`global:result` will be accessible by all scripts and from the command line. Further it will be permanently available until explicitly deleted with `UnLet`. After the variable has been created with the `global:` prefix, it can subsequently be omitted. For example in:

```
Let global:result = 10
Show result
Let result = 11
Show result
```

will display

```
result=10
result=11
```

in the message window. The variable `result` will be available to other scripts whereas if the `global:` prefix had been left off, it would not. Although it is not necessary to include the `global:` prefix except when first creating the variable, it is nevertheless good practice to do so to aid readability of the script.

Empty Values

Many functions return *empty* values (also known as empty vectors) when they are unable to produce a return value. An empty value contains no data. An empty value can be tested with the `Length` function which will return 0. All other functions and operators will yield an error if presented with an empty value.

Empty values should not be confused with empty strings. The latter is explained in the next section.

Empty Strings

An empty string is one that has no characters. An empty string can be entered on a command line with the character sequence:

```
{ '' }
```

Empty strings are not the same as empty values. An empty value has no data at all and will result in an error if supplied to any function other than the Length function.

Quotes: Single and Double

Single quotation marks (') and double quotation marks (") both have a special, but different, meaning in SIMetrix and in the past this has been the source of much confusion. Here we explain what each means and when they should be used.

Single quotes are used to signify a text string in an expression. Expressions are used as arguments to the Plot, Curve, Let and Show commands, they are used in braced substitutions and also as the tests for if, for and while statements. These are the only places where you will find or need single quotes.

Double quotes are used in commands to bind together words separated by spaces or semi-colons so that they are treated as one. Normally spaces and semi-colons have a special meaning in a command. Spaces are used to separate arguments of the command while semi-colons terminate the command and start a new one. If enclosed within double quotes, these special meanings are disabled and the text within the quotes is treated as a single argument to the command. Double quotes are often used to enclose strings that contain spaces (see example) but this doesn't necessarily have to be the case.

Examples

```
Let PULSE_SPEC = 'Pulse 0 5 0 10n 10n 1u 2.5u'
```

In the above line we are assigning the variable PULSE_SPEC with a string. This is an expression so the string is in single quotes. Let is a command but it is one of the four commands that take an expression as its argument.

```
Prop value "Pulse 0 5 0 10n 10n 1u 2.5u"
```

Prop is a command that takes a number of arguments. The second argument is the value of a property that is to be modified. In the above line, the new property value, Pulse 0 5 0 10n 10n 1u 2.5u has spaces in it so we must enclose it double quotation marks so that the command treats it as a single string. If there were no quotes, the second argument would be just Pulse and the remainder of the line would be ignored. If an argument contains no spaces or semi-colons then no quotes are necessary although they will do no harm if present.

Where you need both single and double quotes

There are situations where both single and double quotes are needed together. In some of the internal scripts you will find the `scan` function ([page 278](#)) used to split a number of text strings separated by semi-colons. The second argument to `scan` is a string and must be enclosed in single quotation marks. But this argument is also a semi-colon which, despite being enclosed in single quotes, will still be recognised by the command line interpreter as an end-of-command character. So this must be enclosed in double quotes. The whole expression can be enclosed in double quotes in this case.

If you need a literal quote

If you need a string that contains a double or single quote character, use two of them together.

Expressions

An expression is a sequence of variable *names*, *constants*, *operators* and *functions* that can be evaluated to yield a result. Expressions are required by four commands: `Let`, `Curve`, `Plot` and `Show` and they are also used in *braced substitutions* (see [page 32](#)) and *if statements*, *while statements* and *for statements*. This section describes expression syntax and how they are evaluated.

Examples

```
x+2
(v1_p-vout)*r1#p
idx<15
vout[23]-vout[22]
'Hello ' & 'World'
val = PropValues('value', 'ref', refs[idx])
```

The last in the above example is an *assignment expression* and is a special case. These are explained below.

Operators

Loosely, expressions are constants, variables or/and function calls separated by operators. Available operators are:

Arithmetic:

```
+ - * / ^ %
```

'%' performs a remainder function

Relational

```
< > == <= >=
```

Important: a single '=' can be used as equality operator if used in an *if* or *while* statement. In other places it is an assignment operator and '==' must be used for equality.

Logical

AND, OR, NOT,
&& || !

Note: AND, OR, NOT are equivalent to && || ! respectively.

String

&

'&' concatenates two strings.

Operator precedence

When calculating an expression like 3+4*5, the 4 is multiplied by 5 first then added to 3. The multiplication operator - '*' - is said to have higher precedence then the addition operator - '+'. The following lists all the operators in order of precedence.

() []
Unary - +¹ NOT !
^
* / %
+ -
< > <= >= ==
AND &&
OR ||
&
=
,

Notes

1. A single '=' is interpreted as '==' meaning equality when used in *if statements* and *while statements* and has the same precedence.
2. Parentheses have the highest precedence and are used in their traditional role to change order of evaluation. So (3+4)*5 is 35 whereas 3+4*5 is 23.
3. The comma ',' is used as a separator and so has the lowest precedence.

Functions

Functions are central to SIMetrix scripts. All functions return a value and take zero or more *arguments*. The `sqrt` function for example takes a single argument and returns its square root. So:

Let x = sqrt(16)

will assign 4 to x.

Functions are of the form:

-
1. E.g. In 5*-3, the '-' is a unary operator - applying to a single value not operating on two values. In this instance '-' has higher precedence than '*'.

```
function_name( [ argument, ... ] )
```

Examples

Function taking no arguments:

```
NetName ()
```

function taking two arguments:

```
FFT( vout, 'Hanning')
```

Functions don't just perform mathematical operations like square root. There are functions for string processing, functions which return information about some element of the program such as a schematic or graph, and there are user interface functions. Complete documentation on all available functions is given in [“Function Reference” on page 73](#)

Braced Substitutions

A braced substitution is an expression enclosed in curly braces '{' and '}'. When the script interpreter encounters a braced substitution, it evaluates the expression and substitutes the expression and the braces with the result of the evaluation - as if it had been typed in by the user. Braced substitutions are important because, with the exception of `Let`, `Show`, `Plot` and `Curve`, commands cannot accept expressions as arguments. For example, the `Echo` command displays in the message window the text following the `Echo`. If the command `Echo x+2` was executed, the message `x+2` would be displayed not the result of evaluating `x+2`. If instead the command was `Echo { x+2 }` the result of evaluating `x+2` would be displayed.

If the expression inside the braces evaluates to a vector each element of the vector will be substituted. Note that the line length for commands is limited (although the limit is large - in excess of 2000 characters) so substituting vectors should be avoided unless it is known that the vector does not have many elements.

Braced substitutions may not be used in the control expression for conditional statements, while loops and for loops. For example, the following is not permitted

```
if {netname()} < 4.56 then
```

To achieve the same result the result of the braced expression must be assigned to a variable e.g.:

```
let v = {netname()}  
if v < 4.56 then
```

Bracketed Lists

These are of the form

```
[ expression1, expression2, ... ]
```

The result of a bracketed list is a vector of length equal to the number of expressions separated by commas. There must be at least one expression in a bracketed list - an empty list is not permitted. For example:

```
Let v = [3, 5, 7]
```

assigns a vector of length 3 to v . So $v[0]=3$, $v[1]=5$ and $v[2]=7$. The expressions in a bracketed list may be any type, as long they are all the same. The following for example, is illegal:

```
Let v = [3, 'Hello', 'World']
```

The second element is of type string whereas the first is real. The following example is however legal:

```
Let v = ['3', 'Hello', 'World']
```

3 which is real has been replaced by '3' which is a string.

Type Conversion

Most functions and operators expect their arguments to be of a particular type. For example the $+$ operator expects each side to be a numeric (real or complex) type and not a string. Conversely, the $\&$ operator which concatenates strings naturally expects a string on each side. The majority of functions also expect a particular type as arguments, although there are some that can accept any type.

In the event that the type presented is wrong, SIMetrix will attempt to convert the value presented to the correct type. To convert a numeric value to a string is straightforward, the value is simply represented in ASCII form to a reasonable precision. When a string is presented but a numeric value is required, the string is treated as if it were an expression and is evaluated. If the evaluation is successful and resolves to the correct type the result is used as the argument to the operator or function. If the evaluation fails for any reason an error message will be displayed.

Aliases

An *alias* is a special type of string. Alias strings hold an expression which is always evaluated when used. The simulator outputs some of its data in alias form to save memory and simulation time. For example, the currents into subcircuit pins are calculated by adding the currents of all devices within the subcircuit connected to that pin. If its efficient to do so, this current is not calculated during simulation. Instead the expression to perform that calculation is stored as an alias so that it can be calculated if needed. Aliases may also be created using the `MakeAlias` command see [page 386](#).

Statements and Commands

Scripts are composed of a sequence of *statements*. Statements usually comprise at least one *command* and optionally control words such as `if` and `then`. A *command* is a single line of text starting with one of the 380 or so command names listed in the “[Command Reference](#)”.

There are six types of statement. These are:

command statement
if statement
while statement
for statement
jump statement
script statement

Commands

Commands begin with one of the names of commands listed on [page 327](#). A command performs an action such as running a simulation or plotting a result. E.g.:

```
Plot v1_p
```

is a command that will create a graph of the vector `v1_p`. The syntax varies for each command. Full details are given in the “[Command Reference](#)”.

All commands must start on a new line or after a semi-colon. They must also end with a new line or semi-colon.

A command statement is a sequence of one or more commands.

Command Switches

Many commands have *switches*. These are always preceded by a `/` and their meaning is specific to the command. There are however four global switches which can be applied to any command. These *must* always be placed immediately after the command. Global switches are as follows:

- `/e` Forces command text to copied to command history. Use this when calling a command from a script that you wish to be placed in the command history
- `/ne` Inhibits command text copying to command history. Use this for commands executed from a menu or key definition that you do *not* wish to be included in the command history.
- `/quiet` Inhibits error messages for that command. This only stops error message being displayed. A script will still be aborted if an error occurs but no message will be output
- `/noerr` Stops scripts being aborted if there is an error. The error message will still be displayed

If Statement

An *if statement* is of the form:

```
if expression then  
    statement  
endif
```

OR

if *expression* **then**

statement

else

statement

endif

OR

if *expression* **then**

statement

[[**elseif** *expression* **then**

statement]...]

else

statement

endif

Examples

```

if NOT SelSchem() then
    echo There are no schematics open
    exit all
endif

if length(val)==1 then
    echo {refs[idx]} {val}
else
    echo Duplicate reference {refs[idx]}. Ignoring
endif

if opts[0] && opts[1] then
    let sel = 1
elseif opts[0] then
    let sel = 2
else
    let sel = 3
endif

```

In form1, if the expression resolves to a TRUE value the statement will be executed. (TRUE means not zero, FALSE means zero). In the second form the same happens but if the expression is FALSE the statement after the `else` is executed. In the third form, if the first expression is FALSE, the expression after the `elseif` is tested. If that expression is TRUE the next statement is executed if not control continues to the next `elseif` or `else`.

While Statement

While statements are of the form:

```

do while expression
    statement

```

loop

OR (alternative form)

while *expression*

statement

endwhile

Example

```
do while GetOption(opt) <> 'FALSE'
  let n = n+1
  let opt = 'LibFile' & (n+99)
loop
```

Both forms are equivalent.

In while loops the expression is evaluated and if it is TRUE the statement is executed. The expression is then tested again and the process repeated. When the expression is FALSE the loop is terminated and control passes to the statement following the endwhile.

For Statement

These are of the form:

for *variable=expression1 to expression2* [**step constant**]

statement

next *variable*

Example

This finds the sum of all the values in array.

```
for idx=0 to length(array)-1
  let sum = sum + array[idx]
next idx
```

A for loop executes *statement* for values of *variable* starting at *expression1* and ending with *expression2*. Each time around the loop variable is incremented by *expression3* or if there is no step expression, by 1. If *expression2* starts off with a value less than *expression1*, *statement* will not be executed at all.

Script Statement

A script statement is a call to execute another script. Scripts are executed initially by typing their name at the command line (or if the script has .sxscr extension, the .sxscr can be omitted) or selecting a key or menu which is defined to do the same. Scripts can also be called from within scripts in which case the call is referred to as *script statement*. Note that a script may not call itself.

Exit Statement

There are four types:

exit while

exit for

exit script

exit all

`exit while` forces the innermost while loop to terminate immediately. Control will pass to the first statement after the terminating `endwhile` or `loop`.

`exit for` does the same for for-loops.

`exit script` will force the current script to terminate. Control will pass to the statement following the call to the current script.

`exit all` will abort all script execution and control will return to the command line.

Accessing Simulation Data

Overview

When a simulation is run, a number of vectors (scalars for dc operating point) are created providing the node voltages and branch currents of the circuit. These are just like variables used in a script and can be accessed in the same way. There are however a number of differences from a normal variable. These are as follows:

- Simulation vectors are placed in their own *group*.
- They are usually attached to a *reference* vector.
- They usually have a *physical type* (e.g. Volts, Amps etc.)
- Some are *aliases*. See [page 33](#)

Each of these is described in the following sections.

Groups

All variables are organised into groups. When SIMetrix first starts, there is only one called the *Global* group and all global variables are placed in it. (See “[Scope of Variables, Global Variables](#)” on [page 28](#)). When a script executes a new group is created for it and its own - local - variables are placed there. The group is destroyed when the script exits as are its variables.

Each time a simulation run is started a new group is created and the data generated by the analysis is placed in the group. Groups from earlier runs are not immediately destroyed so that results from earlier runs can be retrieved. By default, three simulation groups are kept at any time with the oldest being purged as new ones are created. A particular group can be prevented from being purged by selecting the menu Graphs and Data|Keep Current Data Group. Further the number of groups kept can be changed

with the `GroupPersistence` option. See the “Sundry Topics” chapter of the *User's Manual* for details about Options.

Groups provide a means of organising data especially simulation data and makes it possible to keep the results of old simulation runs.

All groups have a name. Simulation group names are related to the analysis being performed. E.g. transient analyses are always `trann` where *n* is a number chosen to make the name unique.

Variables within a group may be accessed unambiguously by using their *fully qualified* name. This is of the form:

groupname:variable_name

E.g. `tran1:vout`

The Current Group

At any time a single group is designated the *current* group. This is usually the group containing the most recent simulation data but may be changed by the user with the Graphs and Data|Change Data Group... menu or with the `SetGroup` command. If a variable name is used in an expression that is not local (created in a script) or global, the current group is searched for it. So when the command `Plot vout` is executed if `vout` is not a local or global variable SIMetrix will look for it in the current group.

You can view the variables in the current group with the `Display` command. Run a simulation and after it is completed type `Display` at the command line. A list of available variables from the simulation run will be displayed. Some of them will be *aliases*. These are explained on [page 33](#)

The ':' prefix

If a variable name is prefixed with a colon it tells SIMetrix to only search the current group for that name. Local or global variables of the same name will be ignored.

The colon prefix also has a side effect which makes it possible to access vectors created from numbered nodes. SPICE2 compatible netlists can only use numbers for their node (=net) names. SIMetrix always creates simulation vectors with the same name as the nets. If the net name is a number, so is the variable name. It was stated earlier that variable names must begin with a non-numeric character but in fact this is only partly true. Variable names that start with a digit or indeed consist of only digits can be used but the means of accessing them is restricted. Prefixing with a ':' is one method. The `Vec()` function ([page 317](#)) can also be used for this purpose.

Multi-division Vectors

Multi-step runs such as Monte Carlo produce multiple vectors representing the same physical quantity. In SIMetrix version 3.1 and earlier these vectors remained independent but the groups to which they were attached were bundled together into a *collection*. From version 4 the multiple vectors are in effect joined together into a *multi-division vector*. This is similar to a two dimensional vector (or array or matrix) except that the rows of the matrix are not necessarily all the same length.

When plotting a multi-division vector, each individual vector - or division - will be displayed as a single curve. If listing or printing a multi-division vector with the `Show` command, all the divisions will be listed separately.

You can access a single vector (or division) within a multi-division vector using the index operators - '[' and ']'. Suppose `vOUT` was a multi-division vector with 5 divisions. Each individual vector can be accessed using `vOUT[0]`, `vOUT[1]`, `vOUT[2]`, `vOUT[3]` and `vOUT[4]`. Each of these will behave exactly like a normal single division vector. So, you can use the index operator to access single elements e.g. `vOUT[2][23]` retrieves the single value at index 23 in division 2.

To find the number of divisions in a multi-division vector, use the function [NumDivisions \(page 246\)](#).

You can collate values at a given index across all divisions using the syntax: `vectorname[][index]`. E.g. in the above example `vOUT[][23]` will return a vector of length 5 containing the values of index 23 for all 5 divisions.

Multi-division vectors may be combined using arithmetic operators provided either both sides of the operator are compatible multi-division vectors - i.e. have identical x-values - or one of the values is a scalar.

Multi-division Vectors in Functions

Not all functions accept multi-division vectors for their arguments. The following table lists the functions that do accept multi-division vectors. The entry for each argument specifies whether that argument accepts multi-division vectors and how the data is dealt with.

“X”	Multi-division vectors are not accepted for this argument.
“Scalar”	The function acts on the multi-division vector to obtain a scalar value.
“Vector”	The function obtains a scalar value for each division within the multi-division vector.
“Multi”	The function processes all the vector's data to return a multi-division vector

Function name	Arg 1	Arg 2	Arg 3	Arg 4
<code>abs</code>	Multi			
<code>atan</code>	Multi			
<code>atan_deg</code>	Multi			
<code>cos</code>	Multi			
<code>cos_deg</code>	Multi			
<code>db</code>	Multi			
<code>DefineFourierDialog</code>	X	Scalar		

Function name	Arg 1	Arg 2	Arg 3	Arg 4
diff	Multi			
Execute	X	Multi	Multi	Multi
exp	Multi			
fft	Multi	X		
FIR	Multi	X	X	
Fourier	Multi	X	X	X
FourierOptionsDialog	X	Scalar		
FourierWindow	Multi	X	X	
GetVecStepParameter	Scalar			
GetVecStepVals	Scalar			
GroupDelay	Multi			
Histogram	Multi	X		
IIR	Multi	X	X	
im	Multi			
imag	Multi			
integ	Multi			
Interp	Multi	X	X	X
IsComplex	Scalar			
IsNum	Scalar			
IsStr	Scalar			
Length	Scalar			
In	Multi			
log	Multi			
log10	Multi			
mag	Multi			
magnitude	Multi			
maxidx	Multi			
Maxima	Multi	X	X	
Maximum	Multi	X	X	
mean	Multi			
Mean1	Multi	X	X	
minidx	Multi			
Minima	Multi	X	X	

Function name	Arg 1	Arg 2	Arg 3	Arg 4
Minimum	Multi	X	X	
norm	Multi			
NumDivisions	Scalar			
NumElems	Vector			
ph	Multi			
phase	Multi			
phase_rad	Multi			
PhysType	Scalar			
Range	Multi	X	X	
re	Multi			
real	Multi			
Ref	Multi			
RefName	Scalar			
Rms	Multi			
RMS1	Multi	X	X	
rnd	Multi			
RootSumOfSquares	Multi	X	X	
sign	Multi			
sin	Multi			
sin_deg	Multi			
sqrt	Multi			
SumNoise	Multi	X	X	
tan	Multi			
tan_deg	Multi			
Truncate	Multi	X	X	
Units	Scalar			
Val	Multi			
XFromY	Multi	X	X	X
XY	Multi	Multi		
YFromX	Multi	X	X	

Vector References

Simulation vectors are usually attached to a *reference*. The reference is a vector's x-values. E.g. any vector created from a transient analysis simulation will have a reference of time. AC analysis results have a reference of frequency.

Vectors created by other means may be assigned a reference using the `SetRef` command. See [page 423](#)

Physical Type

Simulation vectors also usually have a *physical type*. This identifies the values units e.g. Volts or Amps. When evaluating expressions SIMetrix attempts to resolve the physical type of the result. For example, if a voltage is multiplied by a current SIMetrix will assign the Physical Type Watts to the result.

Any vector can be assigned a physical type using the `SetUnits` command [page 424](#).

User Interface to Scripts

Dialog Boxes

A number of functions are available which provide means of obtaining user input through dialog boxes. These are:

Function name	Page	Comment
AddRemoveDialog	page 76	Add or remove items to or from a list
BoolSelect	page 79	Up to 6 check boxes.
ChooseDir	page 81	Select a directory
EditObjectPropertiesDialog	page 120	Read/Edit a list of property names and values
EditSelect	page 128	Up to 6 edit boxes
EnterTextDialog	page 131	Enter multi line text
GetSIMetrixFile	page 189	Get file name of pre-defined type
GetUserFile	page 205	Get file name (general purpose)
InputGraph	page 217	Input text for graph
InputSchem	page 217	Input text for schematic
NewValueDialog	page 244	General purpose dialog box.
RadioSelect	page 265	Up to 6 radio buttons
SelectDialog	page 283	Select item(s) from a list

Function name	Page	Comment
TreeListDialog	page 310	Select item from tree structured list
UpDownDialog	page 314	Re order items
UserParametersDialog	page 314	Read/Edit a list of parameter names and values
ValueDialog	page 316	Up to 10 edit boxes for entering values

The above are the general purpose user interface functions. In particular, the function [NewValueDialog \(page 244\)](#) is very universal in nature and has a wide range of applications. There are many more specialised functions. These are listed in [“Functions by Application” on page 67](#)

User Control of Execution

Sometimes it is desirable to have a script free run with actions controlled by a key or menu item. For example you may require the user to select an arbitrary number of nodes on a schematic and then press a key to continue operation of the script to perform - say - some calculations with those nodes. You can use the `DefKey` and `DefMenu` commands to do this. However, for a key or menu to function while a script is executing, you must specify “immediate” mode when defining it. Only a few commands may be used in “immediate” mode definitions. To control script execution, the `Let` ([page 385](#)) command may be used. The procedure is to have the key or menu assign a global variable a particular value which the script can test. The following example outputs messages if F2 or F3 is pressed, and aborts if F4 is pressed.

```
defkey F2 "scriptresume;let global:test=1" 5
defkey F3 "scriptresume;let global:test=2" 5
defkey F4 "scriptresume;let global:test=0" 5

let global:test = -1
while 1
  scriptpause
  if global:test=0 then
    exit script
  elseif global:test=1 then
    echo F2 pressed
  elseif global:test=2 then
    echo F3 pressed
  endif
  let global:test = -1
endwhile

unlet global:test
```

Errors

Loosely, there are two types of error, syntax errors and execution errors.

Syntax Errors

Syntax errors occur when the script presented deviates from the language rules. An `endif` missing from an *if statement* for example. SIMetrix will attempt to find all syntax errors - it won't abort on the first one - but it will not execute the script unless the script is free of syntax errors. Sometimes one error can hide others so that fixing syntax errors can be an iterative process. On many occasions SIMetrix can identify the details of the error but on some occasions it is unable to determine anything other than the fact that it isn't right. In this instance a "Bad Statement" error will be displayed. These are usually caused by unterminated *if*, *while* or *for* statements. Although in many cases SIMetrix can correctly identify an unterminated statement, there are some situations where it can't.

Note that a syntax error in an expression will not be detected until execution.

Execution Errors

These occur when the script executes and are mostly the result of a command execution failure or an expression evaluation failure. Refer to error message documentation in the Help system for details of individual messages.

Error Messages

A listing of virtually all possible error messages is provided in the on-line help. This can be accessed from menu Help|Error Messages.

Executing Scripts

Scripts are executed by typing their file name at the command line or selecting the menu File|Scripts|Run Script.... Additionally, scripts can be assigned to a key or menu. See "User Defined Key and Menu Definitions" on page 434.

If a full pathname is not given, SIMetrix first searches a number of locations. The rules are a little complicated and are as follows:

1. Search the BiScript directory followed by all its descendants. On Windows the BiScript directory is usually at `<simetrix_root>\support\biscript`. On Linux it defaults to `/usr/local/simetrixXX/share/biscript` where *XX* is the version - e.g. 50.
2. Search for a built in script of that name. Built in scripts are bound into the executable binary of SIMetrix. See "Built-in Scripts" on page 46.
3. Search the SCRIPT directory. This is defined by the ScriptDir option setting (see "Set" on page 420) which can also be accessed in the File Locations tab of the options dialog box. (File|Options|General...).
4. Search the User Script list of directories. This is defined by the UserScriptDir option variable (see "Set" on page 420). This may be set to a semi-colon delimited list of search paths.
5. Search the current working directory if the script was executed from a menu or the command line. If the script was called from another script, the directory where the calling script was located is searched instead

Scripts can also be executed using the Execute command. See [page 379](#) for details.

Script Arguments

You can pass data to and from scripts using arguments.

Passing by Value

To pass a value *to* a script, simply place it after the script name. E.g.

```
my_script 10
```

The value 10 will be passed to the script. There are two methods of retrieving this value within the script. The easiest is to use the Arguments command ([page 348](#)). In the script you would place a line like:

```
Arguments num
```

In the above the variable `num` would be assigned the value 10. If the Arguments command is used, it becomes compulsory to pass the argument. If you wish to provide a script with optional arguments you must use the `$arg` variables. When an argument is passed to a script a variable with name `$argn` is assigned with the value where *n* is the position of the argument on the command line starting at 1. To find out if the argument has been passed, use the ExistVec function. E.g.

```
if ExistVec('$arg1') then
  .. action if arg 1 passed
else
  .. action if arg 1 not passed
endif
```

Passing by Reference

When an argument is passed by value, the script in effect obtains a local copy of that data. If it subsequently modifies it, the original data in the calling script remains unchanged even if a variable name was used as the argument. The alternative is to *pass by reference* which provides a means of passing data back to the calling script. To pass by reference you must pass a variable prefixed with the @ character. E.g.

```
Let var = 10
my_script @var
```

To retrieve the value in the called script we use the Arguments command as we did for passing by value but also prefix with @. E.g.

```
Arguments @var
Let var = 20
```

The above modifies `var` to 20 and this change will be passed back to the `var` in the calling script. In the above example we have used the same variable name `var` in both the called and calling scripts. This is not necessary, we have just done it for clarity. You can use any name you like in either script.

Optional arguments passed by reference work the same way as arguments passed by value except that instead of using the variable `$argn` you must use `$varn`. You do not

need to use @ when accessing arguments in this way. See the internal script `define_curve` for an example.

Important

There is currently a limitation that means you can't use an argument passed by reference directly in a braced substitution. E.g.

```
{var}
```

where `var` is an argument passed by reference will not work. Instead you can assign the value to a local variable first.

Passing Large Arrays

In many computer languages it is usually recommended that you pass large data items such as arrays by reference as passing by value involves making a fresh copy which is both time consuming and memory hungry. Passing by reference only passes the location of the data so is much more efficient. In the SIMetrix script language, however, you can efficiently pass large arrays by value as it uses a technique known as *copy on write* that does not make a copy of the data unless it is actually modified.

Built-in Scripts

All the scripts needed for the standard user interface are actually built in to the executable file. The source of all of these can be found on the installation CD.

Debugging Scripts

To see display of commands executed

You can watch the script being executed line by line by typing at the command line before starting the script:

```
Set EchoOn
```

This will cause the text of each command executed to be displayed in the message window. When you have finished you cancel this mode with:

```
Unset EchoOn
```

To single step a script

Run the script by typing at the command line:

```
ScriptPause ; scriptname
```

where *scriptname* is the name of the script you wish to debug. To be useful it is suggested that you enable echo mode as described above. To single step through the script, press F2.

Note that `ScriptPause` only remains in effect for the first script. Subsequent scripts will execute normally.

To abort a currently executing script

Press escape key

To pause a currently executing script

Press shift-F2. Note that it is not possible to run other commands while a script is paused but you can single step through it using F2.

To resume a paused script

Press cntrl-F2

Startup Script

The startup script is executed automatically each time SIMetrix is launched. By default it is called startup.sxscr but this name can be changed with in the options dialog box. (File|Options|General...). The startup file may reside in the script directory (defined by ScriptDir option variable) or in a user script directory (defined by UserScriptDir option variable).

The most common use for the startup script is to define custom menus and keys but any commands can be placed there.

To edit the startup script, select the File|Scripts|Edit Startup menu item.

Chapter 3 Function Summary

Function Summary

The following table lists all functions available. Shaded boxes indicate functions that are either new for version 5.5, have been updated or their documentation has been updated..

Function Name	Description
abs (real/complex)	Absolute value
ACSourceDialog (real)	Displays dialog box intended for the user definition of an AC source
AddConfigCollection (string, string)	Adds a list of entries to a named section in the configuration file
AddGraphCrossHair (string)	Adds a new cursor to the current graph
AddModelFiles (string)	Install list of model paths
AddPropertyDialog ([string, string, string])	User interface function. Open add property dialog for symbol editor
AddRemoveDialog (string, string [, string, string])	User interface function. Allows selection of a list of items.
AddSymbolFiles (string)	Adds file or files to list of installed symbol library files
arg (real/complex)	Phase of argument in degrees. Result always between -180 to +180
arg_rad (real/complex)	Phase of argument in radians. Result always between $-\pi$ and $+\pi$
Ascii (string)	Returns ASCII code for character
AssociateModel (string, string [, string, string])	Special purpose function for managing parts browser.
atan (real/complex)	Arc tangent (radians)
atan_deg (real/complex)	Arc tangent (degrees)
BoolSelect ([real, string, string])	User interface function. Returns state of up to 6 check boxes
Branch ()	Returns branch current formula of schematic net nearest cursor
CanOpenFile (string)	Returns TRUE if specified file exists and can be opened for read
ChangeDir (string)	Change current working directory and return value indicating result
Char (string, real)	Returns character from string

Function Name	Description
ChooseDir ([string, string, string])	User interface function. Returns user selected pathname.
ChooseDirectory (string)	User interface function. Returns user selected pathname.
Chr (real)	Returns a string consisting of a single character specified by an ASCII code
CloseEchoFile ()	Closes the file associated with the Echo command. (See also OpenEchoFile)
CloseFile (real)	Closes a file opened using OpenFile
CloseSchematic (real)	Close a schematic handle opened using OpenSchematic
CollateVectors (string, string [, real])	Returns vector data in an interleaved manner
CompareSymbols (string, string)	Compare two schematic symbols
ComposeDigital (string [, real, string, real])	Builds a new vector from a binary weighted combination of digital vectors
ConvertLocalToUnix (string)	Convert file name to UNIX format using '/'
ConvertUnixToLocal (string)	Convert filename to local format using '\ ' on Windows, '/' on Linux
CopyURL (string, string [, string])	Copy a file to or from a location defined by a URL. Supports http, ftp and local files.
cos (real/complex)	Cosine (radians)
cos_deg (real/complex)	Cosine (degrees)
CreateLockFile (string, string)	Create or remove a lock file for specified file.
CreateDiodeDialog ([string])	Opens a specialised dialog used by the diode model in-circuit parameter extractor
CreateShortcut (string, string, string)	(Windows only) Create a shortcut to the specified path
CyclePeriod (real, real [, real, real])	Returns the time between zero crossing pairs with the same slope direction. It can be used for plotting frequency vs time
CreateTimer (string, real [, string])	Create a timer to schedule events in the future
cv	Alias to GetCurveVector
Date ([string])	Return current system date in string form.

Function Name	Description
<code>db(real/complex)</code>	$dB(x) = 20 * \log_{10} (\text{mag}(x))$
<code>DCSourceDialog(real)</code>	Opens 'Edit DC Source' dialog box
<code>DefineADCDialo(real)</code>	UI function to define generic ADC
<code>DefineArbSourceDialog(string)</code>	UI function to define arbitrary source
<code>DefineBusPlotDialog(string [string,])</code>	Opens a dialog box to allow the user to plot a bus
<code>DefineCounterDialog(real)</code>	UI function to define generic counter
<code>DefineCurveDialog(string)</code>	UI function, opens define curve dialog.
<code>DefineDACDialog(real)</code>	UI function to define generic DAC
<code>DefineFourierDialog(string [, real])</code>	UI function, opens define fourier dialog
<code>DefineIdealTxDialog(real, real, real [, string])</code>	UI function to define ideal transformer
<code>DefineLaplaceDialog(string)</code>	UI function to define S-domain transfer function
<code>DefineLogicGateDialog(real)</code>	UI function to define generic logic gate
<code>DefinePerfAnalysisDialog(string)</code>	UI function for defining a performance analysis
<code>DefineRegisterDialog(real)</code>	UI function to define Bus register
<code>DefineRipperDialog(string, string)</code>	UI function to define schematic bus ripper
<code>DefineSaturableTxDialog(string, string, real, real)</code>	Open dialog box to define a saturable transformer
<code>DefineShiftRegDialog(real)</code>	UI function to define generic shift register
<code>DefineSimplisMultiStepDialog</code>	Open dialog box to define SIMPLIS multi-step dialog.
<code>DeleteConfigCollection(string)</code>	Deletes a list of entries in the config file
<code>DeleteTimer(real)</code>	Deletes a timer
<code>DescendDirectories(string)</code>	Returns all directories under the specified directory, recursing through all sub-directories
<code>DescendHierarchy([string, real])</code>	Analyse schematic hierarchy
<code>diff(real)</code>	Return derivative of argument
<code>EditArcDialog([real, real])</code>	UI function to edit symbol editor arc
<code>EditAxisDialog(string)</code>	UI function, opens edit axis dialog
<code>EditBodePlotProbeDialog(string)</code>	UI function for editing Bode plot fixed probes

Function Name	Description
EditCrosshairDimensionDialog (string, string [, string])	UI function, opens dialog for editing cursor dimension
EditCurveMarkerDialog (string, string [, string])	UI function, opens dialog to edit curve marker
EditDeviceDialog (string, string [, string, string])	UI function to select device and edit device parameters
EditDigInitDialog (real)	UI function to edit digital initial condition
EditFreeTextDialog (string, string [, string])	UI function, opens dialog to edit graph free text object
EditGraphTextBoxDialog (string, string [, string])	UI function, opens dialog to edit graph text box object
EditLegendBoxDialog (string, string [, string])	UI function, opens dialog to edit graph legend box object
EditObjectPropertiesDialog (string, string [, string, string])	UI function, opens dialog to edit property values
EditPinDialog (string [, string])	UI function to edit symbol editor pins
EditPotDialog (real)	UI function to edit potentiometer properties
EditProbeDialog (string)	UI function, opens edit fixed probe dialog
EditPropertyDialog (string [, string, string, string])	UI function to edit symbol editor properties
EditReactiveDialog (string, string [, string, string])	Opens a dialog box designed to edit inductors and capacitors
EditSelect ([string, string, string])	User interface function. Returns entries in up to 6 edit controls
EditTimer (real, string [, real])	Edit a timer
EditWaveformDialog (real [,real])	Opens the dialog box editing a time domain waveform
EnterTextDialog (string)	UI function to define multi line text
EscapeString (string)	Process string and replace escaped characters with literals
ev (any [, any, any, any, any, any, any, any])	Special function used to evaluate a sequence of expressions
Execute (string [, any, any, any])	Execute script as a function
ExistDir (string)	Returns TRUE if specified directory exists
ExistFunction (string [, string])	Returns TRUE if the specified function exists.

Function Name	Description
ExistSymbol (string [, string])	Returns TRUE if specified schematic symbol exists.
ExistVec (string [, string])	Returns TRUE if specified variable name exists.
exp (real/complex)	Exponential
fft (real [, string])	Fast Fourier Transform
Field (real, real, real)	Provides bit-wise access to integers
FilterFile (string, string [,string])	Unsupported function. Filters specific lines from a text file.
FindModel (string, string [,string])	Returns location of device model given name and type
FIR (real, real [, real])	Finite Impulse Response digital filter
Floor (real)	Returns argument truncated to next lowest integer
Floorv (real)	As Floor but accepts vector inputs
FormatNumber (real, real [, string])	Returns formatted number in string form
Fourier (real, real, real [, real])	Calculate continuous fourier
FourierOptionsDialog (string [, real])	UI function, opens fourier options dialog
FourierWindow (real [, string, real])	Apply window function for fourier analysis
FullPath (string [, string])	Returns full path name of given relative path
GenPrintDialog (string [, string])	UI function, opens print dialog box
GetActiveWindow ()	Return details about currently active window
GetAllCurves ()	Returns array of curve indexes for all curves in current graph
GetAllSymbolPropertyNames ([string])	Finds names of all the properties on currently open symbol
GetAllYAxes ()	Returns array of axis id's for all y axes in current graph
GetAnalysisInfo ([string])	Return information about most recent analysis
GetAnalysisLines ()	Returns the analysis lines used in the most recent simulation analysis
GetAxisCurves (string)	Returns array of curve id's for all curves attached to specified axis

Function Name	Description
GetAxisLimits (string)	Returns min and max limits and axis type (log or lin) of specified axis
GetAxisType (string)	Returns type (X, Y, Digital etc.) of specified axis
GetAxisUnits (string)	Returns units of specified axis
GetChildModulePorts (string, string [, real])	Finds information about module ports in the underlying schematic of a hierarchical block
GetColours ()	Return names of all colour objects
GetColourSpec (string)	Return specification for a colour object
GetConfigLoc ()	Return location of config information
GetConnectedPins (real [, string, string])	Returns instance and pin name for all instances connected to net at specified point
GetConvergenceInfo ()	Return convergence data for most recent simulation
GetCurDir ()	Return current working directory
GetCurrentGraph ()	Return ID of the currently selected graph
GetCursorCurve ()	Returns curve id and source group name of curve attached to measurement cursor
GetCurveAxis (string)	Returns axis id of specified curve
GetCurveName (string)	Returns name of specified curve
GetCurves ()	Returns curve names in selected graph
GetCurveVector (real [, real, string])	Returns data associated with a graph curve
GetDatumCurve ()	Returns curve id and source group name of curve attached to reference cursor
GetDeviceDefinition (string, string [, string, string])	Retrieve the text of a model definition from library
GetDeviceInfo (string [,string])	Returns information about the specified simulator device
GetDeviceParameterNames (string [, real, string])	Returns list of device parameter names for specified SPICE device
GetDotParamNames ()	Returns names of .PARAM variables used in latest simulation
GetDotParamValue (string)	Returns value of specified .PARAM value in latest simulation run

Function Name	Description
GetDriveType (string)	Determines the type of drive or file system of the specified path
GetEmbeddedFileName (string)	Returns the actual file name used for an embedded file specified using '.FILE' and '.ENDF'
GetEnvVar (string)	Return specified system environment variable
GetEthernetAddresses	Returns information about the installed Ethernet adapters
GetF11Lines (string [, real])	Returns the contents of the schematic's text window (also known as the F11 window)
GetFile (string [, real/complex])	User interface function. Returns user selected file name
GetFileCd (string [, real/complex])	User interface function. As GetFile but changes directory.
GetFileExtensions (string)	Returns file extensions for specified SIMatrix file type
GetFileInfo	Returns information about a specified file
GetFileSave (string)	User interface function. Returns user selected file name for saving
GetFonts ()	Return names of all font objects
GetFontSpec (string)	Return spec. for named font
GetFreeDiskSpace (string)	Returns space available on specified disk volume
GetGraphObjects ([string, string])	Return IDs for specified graph objects
GetGraphObjPropNames (string)	Return property names for specified graph object
GetGraphObjPropValue (string [, string])	Return value for a graph object property
GetGraphTabs ([real])	Return graph ids for graph tabbed sheets
GetGraphTitle ()	Return current graph title
GetGroupInfo (string)	Returns information about a group
GetGroupStepParameter ([string])	Returns the name of the 'stepped parameter' of a multi-step run
GetGroupStepVals ([string])	Returns the 'stepped values' in a multi-step run

Function Name	Description
GetInstanceBounds (string, string [, string])	Returns the bounds occupied by a schematic instance
GetInstanceParamValues (string [, string, string])	Returns parameter values for a simulator device
GetInstancePinLocs ([string, string, string, real])	Returns pin locations of specified instance
GetInstsAtPoint (real, string)	Returns instances at specified point
GetInternalDeviceName (string)	Finds the simulator's internal device name for a model
GetKeyDefs	Returns details of all key definitions created using DefKey
GetLastCommand (string)	Retrieve last command issued by a menu or toolbar
GetLastError ()	Returns result of most recent command
GetLegendProperties (string [, string])	Returns array of legend property
GetLibraryModels (string [, string])	Returns a string array containing information about each model in the specified model library
GetLicenseInfo ()	Returns information about the current license
GetLicenseStats ()	Returns information about the license check out process
GetLongPathName (string)	Returns long path name for path specified either as a long or short path
GetMenuItems (string [, string])	Returns all menu item names in the specified menu
GetModelFiles ()	Return list of installed model files
GetModelName (string)	Returns the model name used by a simulator device
GetModelParameterNames (string [, string])	Returns the names of all real valued parameters of a simulator device model
GetModelParameterValues (string [,string])	Returns the values of all parameters of a simulator model
GetModelType (string)	Returns the simulator internal device name given a user model name
GetModifiedStatus ([real])	Returns modified status of specified schematic
GetNamedSymbolPins (string [, string])	Returns the names for all the pins of a symbol or hierarchical component

Function Name	Description
GetNamedSymbolPropNames (string [, string])	Returns names of all properties defined for a library symbol
GetNamedSymbolPropValue (string, string [, string])	Returns the value of a property defined for a library symbol
GetNearestNet ()	Returns information about the schematic net nearest the mouse cursor
GetNonDefaultOptions ()	Returns names of all explicit .OPTION settings in the most recent simulation
GetNumCurves (string)	Returns number of curves in curve group
GetOpenSchematics ([real])	Returns the path names of all open schematics
GetOption (string)	Returns value of specified option
GetPath (string)	Returns application path
GetPlatformFeatures ()	Returns information on the availability of some platform dependent features
GetPrinterInfo ()	Returns information on installed printers
GetPrintValues ()	Returns the names of all quantities specified in .PRINT controls in the most recent simulation
GetReadOnlyStatus ([real])	Returns internal read-only status of specified schematic
GetSchematicVersion ()	Returns version information about the current schematic
GetSchemTitle ()	Return title of schematic
GetSelectedCurves ([string])	Returns array of curve id's for curves selected in current graph
GetSelectedGraphAnno ()	Return ID of selected graph annotation object
GetSelectedYAxis ()	Returns id of selected Y-Axis
GetShortPathName	Returns short path name for path specified either as a long or short path
GetSimConfigLoc ()	Returns the location of the simulator's configuration information
GetSimetrixFile (string [, string, string])	Returns path name of user selected file
GetSimplisAbortStatus ()	Determines whether a request to abort a SIMPLIS run has been received

Function Name	Description
GetSimplisExitCode()	Returns the application exit code for the most recent SIMPLIS run
GetSimulationErrors()	Retrieves the error messages raised by the most recent simulation run
GetSimulationInfo()	Returns information about the most recent simulation
GetSimulationSeeds	Returns the seeds used for the most recent run
GetSimulatorEvents()	Return list of events for most recent simulation
GetSimulatorMode()	Returns the simulator mode of the current schematic
GetSimulatorOption(string)	Returns the value of a simulator option as used by the most recent analysis
GetSimulatorStats()	Returns statistical information about the most recent run
GetSimulatorStatus()	Returns the current status of the simulator
GetSoaDefinitions()	Returns all Safe Operating Area definitions specified in the most recent analysis
GetSoaMaxMinResults()	Returns the maximum and minimum values reached for all SOA definitions
GetSoaOverloadResults()	Returns the overload factor for each SOA definition
GetSoaResults()	Returns the SOA results for the most recent simulation
GetSymbolArcInfo()	Returns information on symbol editor arc
GetSymbolFiles()	Returns full paths of all installed symbol library files
GetSymbolInfo()	Returns information on symbol editor symbol
GetSymbolOrigin()	Returns the location of the symbol editor's symbol origin point
GetSymbolPropertyInfo([string])	Returns information symbol editor symbol properties
GetSymbolPropertyNames()	Returns symbol editor symbol property names
GetSymbols([string, string])	Returns array of available schematic symbols

Function Name	Description
GetSymbolText (string [, string])	Returns symbol definition in "Compact Text Format"
GetSystemInfo ()	Returns information about the user's system
GetToolButtons ([string])	Returns name and description for available tool buttons
GetUncPath (string)	Returns UNC path of specified path
GetUserFile (string [, string, string, string])	Returns path name of user specified file. Supersedes <code>getfile</code> , <code>getfilecd</code> , and <code>getfilesave</code>
GetVecStepParameter (real/complex)	Returns parameter name associated with vector
GetVecStepVals (real/complex)	Returns parameter values associated with vector
GetWindowNames ([string])	Returns names of current SIMetrix windows
GetXAxis ()	Returns id of x-axis for currently selected graph
GraphLimits ()	Returns x and y limits of selected graph
GroupDelay (real/complex)	Returns group delay of argument
Groups ([string])	Returns array of available groups.
HaveInternalClipboardData (string)	Returns the number of items in the specified internal clipboard
HasLogSpacing (real)	Performs a simple test to determine whether the supplied vector is logarithmically spaced
HasProperty (string [, string, string, real])	Returns true if selected schematic instance has specified property
HighlightedNets ([real])	Returns names for any wholly highlighted net names on the specified schematic
Histogram (real, real [, string])	Returns histogram of argument
Iff (real, any, any)	Returns a specified value depending on the outcome of a test
IIR (real, real [, real])	Infinite Impulse Response digital filter
im (real/complex)	Return imaginary part of argument
imag (real/complex)	same as <code>im()</code>
InputGraph ([string, string])	User Interface function. Input text for graph operation

Function Name	Description
InputSchem ([string, string])	User Interface function. Input text for schematic operation
Instances (string [, real])	Returns array of instances possessing specified property
InstNets ([string])	Returns array of net names for each pin of selected schematic instance
InstNets2 (real, string, string [, string])	As InstNets2 but with more advanced features to identify instance
InstPins ([string, string, real])	Returns array of pin names for each pin of selected schematic instance
InstPoints ([string, string, real])	Returns location and orientation of specified instance
InstProps ([real, string, string])	Returns names of all properties owned by selected instance
Integ (real)	Returns integral of argument
Interp (real, real [, real, real])	Interpolates argument to specified number of evenly spaced points
IsComplex (any)	Returns TRUE if argument is complex
IsComponent (string, string)	Determines whether a schematic instance is a hierarchical component
IsFullPath (string)	Returns TRUE if the supplied path name is a full absolute path
IsModelFile (string [, string])	Determines if a file contains valid electrical models
IsNum (any)	Returns TRUE if argument is numeric (real or complex)
IsSameFile	Compares two paths and returns true (1) if they point to the same file
IsScript (string)	Determines whether the supplied script name can be located
IsStr (any)	Returns TRUE if argument is a string
JoinStringArray (string, string)	Concatenates two string arrays to return a single array
Length (any)	Returns number of elements in vector.
ListDirectory (string [, string])	Returns file names found in a directory matching a supplied wildcard spec
ln (real/complex)	Natural logarithm
Locate (real, real)	Locates value in a monotonic vector. Returns index.
log (real/complex)	Natural logarithm

Function Name	Description
log10 (real/complex)	Base 10 logarithm
mag (real/complex)	Magnitude (same as abs())
magnitude (real/complex)	As mag()
MakeDir (string)	Make a directory and result of operation
MakeLogicalPath (string)	Converts a file system path to a symbolic path
MakeString (real [, string])	Create a string array with specified number of elements
ManageMeasureDialog (string)	Opens dialog box used to manage graph measurements.
Max (real/complex, real/complex)	Returns max of two vectors
Maxidx (real/complex)	Returns index of vector where largest value is held
Maxima (real [, real, string])	Returns locations of maxima of specified vector
Maximum (real/complex [, real, real])	Returns most positive value in vector
mean (real/complex)	Returns statistical mean of all values in vector
Mean1 (real [, real, real])	Returns mean of data in given range
MeasureDialog ([string, string, string])	Opens dialog for specifying graph measurements
MessageBox (string [, string])	Opens a dialog box with a message and user options
Mid (string, real [, real])	Returns substring of string
Min (real/complex, real/complex)	Returns min of two vectors
Minidx (real/complex)	Returns index of vector where smallest value is held
Minima (real [, real, string])	Returns locations of minima of specified vector
Minimum (real/complex [, real, real])	Returns most negative value in vector
ModelLibsChanged ()	Returns TRUE if any installed model paths have changed
Navigate (string, string)	Returns path name of hierarchical block given root path and full component reference.
NearestInst (string)	Cross probe function. Returns nearest schematic instance to cursor

Function Name	Description
NetName ([string])	Cross probe function. Returns the net name of the nearest wire or instance pin.
NetNames ([real])	Returns array of all net names in selected schematic
NetWires (string [, real])	Return all wires on specified net
NewPassiveDialog (string [, string, string, string])	UI function to select passive component value and parameters
NewValueDialog (string, string [, string])	General purpose user input function. Opens a user configurable dialog box
norm (real/complex)	Returns argument scaled so that its largest value is unity.
NumDivisions (real/complex)	Returns number of divisions in a vector
NumElems (any)	Returns number of elements in a vector
OpenEchoFile (string, string)	Redirects the output of the Echo command
OpenFile	Opens a file and returns its handle. This may be used by the Echo command
OpenSchem	Opens a schematic and returns value indicating success or otherwise
OpenSchematic (string)	Opens a schematic without displaying it. Returned 'handle' useable by various functions
Parse (string [, string, string])	Splits string into substrings.
ParseAnalysis (string)	Opens the choose analysis dialog
ParseParameterString (string, string, string [, string, string])	Parses a string of name-value pairs and performs some specified action on them
ParseSimplisInit (string)	Reads and parses the .init file created by a SIMPLIS run
PathEqual (string, string)	Compares two path names with platform dependent case-sensitivity
ph (real/complex)	Returns phase of argument in degrees
phase (real/complex)	As ph()
phase_rad (real/complex)	As ph() but result always in radians
PhysType (real/complex)	Returns physical type of argument
PinName ([string, string])	Cross probe function. Returns pin name nearest to cursor
Probe ()	Displays probe cursor in schematic and waits for mouse click

Function Name	Description
ProcessingAccelerator	Returns 1.0 if the script being executed was called by activating a menu accelerator
Progress (real [, string])	UI function, opens progress bar
PropFlags (string [, string, string, real])	Returns the attribute flags of a schematic property
PropFlags2 (string [, real, string, string])	As PropFlags but with rearranged arguments
PropValue (string)	Returns value of specified property for selected instance
PropValues (string [, string, string, real])	Returns array of property values
PropValues2 (string [, real, string, string])	As PropValues but with rearranged arguments
PutEnvVar	Write an environment variable
PWLDialog (string [,string, real])	Opens a dialog box designed for editing piece wise linear sources
QueryData (string, string)	Filters a list of data items according to search criteria
RadioSelect ([real, string, string, string])	User interface function. Returns user selection of up to 5 radio buttons
Range (any [, real, real])	Returns range of vector (accepts, real, complex and string)
re (real/complex)	Return real part of argument
ReadClipboard ()	Filters a list of data items according to search criteria
ReadConfigCollection	Returns the contents of an entire section in the configuration file
ReadConfigSetting (string [, string])	Reads a configuration setting
ReadFile (string)	Reads text file and returns contents as an array of strings
ReadIniKey (string, string, string)	Reads a key in an "INI" file
ReadRegSetting (string, string [,string])	Reads a string setting from the windows registry
ReadSchemProp (string)	Read schematic window property value
real (real/complex)	As re()
Ref (real/complex)	Returns reference of argument
RefName (real/complex)	Returns the name of the arguments reference vector

Function Name	Description
RelativePath (string [, string])	Returns a relative path name given a full path and a reference path
RemoveConfigCollection	Removes one or more entries from a configuration file collection
RemoveModelFile (string)	Uninstalls a model path
RemoveSymbolFiles (string)	Uninstalls symbol library files
ResolveGraphTemplate (string, string [, string])	Evaluate template string used by graph object
ResolveTemplate (string, string, string)	Evaluate template string
RestartTranDialog (real)	UI function, opens restart transient dialog
Rms (real)	Returns accumulative RMS value of argument
RMS1 (real [, real, real])	Returns RMS of argument over specified range
rnd (real)	Returns random number
RootSumOfSquares (real [, real, real])	Returns root sum of squares of argument over specified range
SaveSpecialDialog ([string])	Opens the dialog used by the schematic's Save Special... menu
Scan (string [, string, real])	Splits a character delimited string into its components.
ScriptName ()	Return name of currently executing script
Search (string, string, [,string])	Search for a string in a list of strings
Seconds ()	Returns the number of seconds elapsed since January 1, 1970
SearchModels (string)	Special purpose used by library installation. Returns pathnames of SPICE compatible model files
SelectAnalysis ()	Opens choose analysis dialog box. Returns value according to how box closed
SelectColourDialog ([string])	UI function, opens colour selection dialog
SelectColumns (string, real [, string])	Analyses an array of character delimited strings and returns selected values.
SelectCount ([string])	Returns number of selected items on schematic

Function Name	Description
SelectDevice (string [, string, string])	Special function forms part of parts browser system. Takes catalog data as arguments and opens dialog box to select a device.
SelectDialog (string, string)	User interface function. Allows selection of one or more items from list
Select2Dialog ([string, string])	Displays a dialog offering two lists
SelectedProperties ([string])	Returns information about selected properties
SelectedWires ()	Returns handles to selected wires on schematic
SelectFontDialog ([string, string])	UI function, opens select font dialog
SelectRows (string, string [, real, string])	Analyses an array of character delimited strings and returns selected values.
SelectSimplisAnalysis ()	Opens SIMPLIS choose analysis dialog box
SelectSymbolDialog ([string, string])	Opens a dialog box allowing the user to select a schematic symbol from the symbol library
SelGraph ()	Returns TRUE if at least one graph is open
SelSchem ()	Returns TRUE if at least one schematic is open
SetReadOnlyStatus (real [, real])	Sets read-only/writeable status of specified schematic
Shell (string [, string])	Runs an external program and returns its exit code
ShellExecute (string [, string, string, string])	Performs an operation on a windows registered file
sign (real)	Return sign of argument
SimetrixFileInfo (string)	Returns information about a SIMetrix file
SimulationHasErrors ()	Determine success of most recent simulation
sin (real/complex)	Sine (radians)
sin_deg (real/complex)	Sine (degrees)
Sleep (real)	Executes a timed delay
Sort (string [, string])	Performs alphanumeric sort on argument.

Function Name	Description
SortIdx (any [, string])	Sorts any vector and returns index order
SourceDialog ([string])	User Interface function. Opens source dialog box for specifying of voltage and current source. Returns string with user selected values
SplitPath (string)	Splits file system path into its components
SprintfNumber (string [,real/complex...])	Print formatted string
sqrt (real/complex)	Square root
Str (string)	Converts argument to string
StringLength (string)	Returns length of string
StrStr (string, string [, real])	Locates a sub string within a string
SubstChar (string, string, string)	Substitutes characters in string
SubstString (string, string, string [, string])	Replaces a substring in a string
SumNoise (real [, real, real])	Same as RootSumOfSquares
SymbolGen (string [, string, string, string])	Special function. Opens symbol generator dialog box. Returns symbol definition in "Compact Text Format"
SymbolInfoDialog ([string, string])	Returns name of schematic symbol
SymbolLibraryManagerDialog ()	Opens the Symbol Library Manager dialog box
SymbolName (string, string [, real])	Returns symbol name of specified instance
SymbolNames ([real, string, string])	Returns symbol names of schematic instances
SymbolPinOrder ([string])	Set and/or return pin order of symbol editor symbol
SystemValue (string)	Returns the value of a system defined variable
TableDialog (real [, string])	Displays a spreadsheet style table to allow the user to enter tabular data
tan (real/complex)	Tangent (radians)
tan_deg (real/complex)	Tangent (degrees)
TemplateGetPropValue (string, string)	Function returns the value of a property. For use in template scripts only
TemplateResolve (string, string)	Resolve TEMPLATE value. For use in template scripts only

Function Name	Description
Time ([string])	Return system time as string
TransformerDialog (string, string, real)	Special function to select transformer characteristics.
TranslateLogicalPath (string)	Converts symbolic path to a physical path
TreeListDialog (string [, string])	General purpose UI function. Open dialog box with tree list control
TRUE (string [, string])	Returns 1 if vector exists and is non-zero
Truncate (real [, real, real])	Returns vector that is a sub range of supplied vector
Units (any)	Returns physical units of argument
unitvec (real)	Returns vector of specified length whose elements are all 1
UpDownDialog (string [, string])	General purpose UI function. Opens dialog with up-down list to allow rearranging order
UserParametersDialog (string, string [, string])	UI function, opens dialog allowing editing of user parameter values
Val (real/complex)	Converts argument to value
ValueDialog ([real, string, string, string])	User interface function. Opens dialog with up to 10 boxes for entering numeric values. Return array of user selected values
Vec (string [, string])	Returns data for named vector. (Allows access to vectors with invalid names)
vector (real)	Returns vector of specified length with each element equal to its index
VectorsInGroup ([string, string])	Returns array of variable names belonging to specified group
VersionInfo ()	Returns version information about running copy of SIMetrix
WirePoints (string [, real])	Returns location of specified wire
Wires ([real])	Return all wires in schematic
WriteConfigSetting (string, string [, string])	Writes a configuration setting
WriteF11Lines (string)	Writes lines directly to the F11 window overwriting any existing lines
WritelnKey (string, string, string [, string])	Writes a key value to an 'INI' file

Function Name	Description
WriteRawData (real/complex, string [, string, string])	Writes data to the specified file in a SPICE3 raw file compatible format
WriteRegSetting (string, string, string [, string])	Writes a string value to the windows registry
WriteSchemProp (string, string [, string])	Write schematic window property value
XCursor ()	Returns x location of graph cursor
XDatum ()	Returns x location of graph reference cursor
XFromY (real, real [, real, real])	Returns array of values specifying horizontal locations where specified vector crosses given y value
XY (real, real)	Creates an XY Vector from two separate vectors
YCursor ()	Returns y location of graph cursor
YDatum ()	Returns x location of graph reference cursor
YFromX (real, real [, real])	Returns array of values specifying the vertical value of the specified vector at the given x value

Functions by Application

Shaded items are new for this version

User Interface

ACSourceDialog	EditCrosshairDimensionDialog	InputGraph
AddPropertyDialog	EditCurveMarkerDialog	InputSchem
AddRemoveDialog	EditDeviceDialog	ManageMeasureDialog
BoolSelect	EditDigInItDialog	MeasureDialog
CreateDiodeDialog	EditFreeTextDialog	MessageBox
DCSourceDialog	EditGraphTextBoxDialog	NewPassiveDialog
DefineADCDialg	EditLegendBoxDialog	NewValueDialog
DefineArbSourceDialog	EditObjectPropertiesDialog	PWLDialg
DefineBusPlotDialog	EditPinDialog	RadioSelect
DefineCounterDialog	EditPotDialog	RestartTranDialog
DefineCurveDialog	EditProbeDialog	SaveSpecialDialog
DefineDACDialog	EditPropertyDialog	SelectColourDialog

User Interface

DefineFourierDialog	EditSelect	SelectDevice
DefineIdealTxDialog	EditReactiveDialog	SelectDialog
DefineLaplaceDialog	EnterTextDialog	Select2Dialog
DefineLogicGateDialog	EditWaveformDialog	SelectFontDialog
DefinePerfAnalysisDialog	FourierOptionsDialog	SelectSymbolDialog
DefineRegisterDialog	GenPrintDialog	SourceDialog
DefineRipperDialog	GetFile	SymbolGen
DefineSaturableTxDialog	GetFileCd	SymbolLibraryManagerDialog
DefineShiftRegDialog	GetFileSave	TableDialog
DefineSimplisMultiStepDialog	GetLastCommand	TransformerDialog
EditArcDialog	GetMenuItems	TreeListDialog
EditAxisDialog	GetSimatrixFile	UpDownDialog
EditBodePlotProbeDialog	GetToolButtons	UserParametersDialog
	GetUserFile	ValueDialog

Mathematical

abs	lff	Minimum
arg	IIR	norm
arg_rad	im	ph
atan	imag	phase
atan_deg	Integ	phase_rad
cos	Interp	re
cos_deg	ln	real
db	log	Rms
diff	log10	RMS1
exp	mag	rnd
fft	magnitude	RootSumOfSquares
Field	Max	sign
FIR	Maxidx	sin
Floor	Maxima	sin_deg
Floorv	Maximum	sqrt
Fourier	mean	SumNoise

Mathematical

FourierWindow	Mean1	tan
GroupDelay	Min	tan_deg
HasLogSpacing	Minidx	unitvec
Histogram	Minima	

Symbol/Schematic

AddSymbolFiles	GetSymbolFiles	PropFlags
Branch	GetSymbolPropertyInfo	PropFlags2
CompareSymbols	GetSymbolPropertyNames	PropValue
CloseSchematic	GetSymbols	PropValues
DescendHierarchy	GetSymbolInfo	Probe
ExistSymbol	GetSymbolOrigin	PropValues2
GetAllSymbolPropertyNames	GetSymbolText	ReadSchemProp
GetChildModulePorts	HasProperty	RemoveSymbolFiles
GetConnectedPins	HighlightedNets	SelectCount
GetF11Lines	Instances	SelectedProperties
GetInstanceBounds	InstNets	SelectedWires
GetInstanceParamValues	InstNets2	SelSchem
GetInstancePinLocs	InstPins	SetReadOnlyStatus
GetInstsAtPoint	InstPoints	SymbolInfoDialog
GetModifiedStatus	InstProps	SymbolName
GetNamedSymbolPins	IsComponent	SymbolNames
GetNamedSymbolPropNames	Navigate	SymbolPinOrder
GetNamedSymbolPropValue	NearestInst	TemplateGetPropValue
GetNearestNet	NetName	TemplateResolve
GetOpenSchematics	NetNames	WirePoints
GetReadOnlyStatus	NetWires	Wires
GetSchematicVersion	OpenSchem	WriteF11Lines
GetSchemTitle	OpenSchematic	WriteSchemProp
GetSymbolArcInfo	PinName	

Graph

AddGraphCrossHair	GetCurves	GetSelectedGraphAnno
GetAllCurves	GetCurveVector	GetSelectedYAxis
GetAllYAxes	GetDatumCurve	GetXAxis
GetAxisCurves	GetGraphObjects	GraphLimits
GetAxisLimits	GetGraphObjPropNames	ResolveGraphTemplate
GetAxisType	GetGraphObjPropValue	SelGraph
GetAxisUnits	GetGraphTabs	XCursor
GetCurrentGraph	GetGraphTitle	XDatum
GetCursorCurve	GetLegendProperties	YCursor
GetCurveAxis	GetNumCurves	YDatum
GetCurveName	GetSelectedCurves	

Vectors/Groups

CollateVectors	Length	Str
ComposeDigital	Locate	TRUE
CyclePeriod	GetGroupInfo	Truncate
ev	GetGroupStepParameter	Units
ExistFunction	GetGroupStepVals	Val
ExistVec	NumDivisions	Vec
GetVecStepParameter	NumElems	vector
GetVecStepVals	PhysType	VectorsInGroup
Groups	Range	WriteRawData
IsComplex	Ref	XFromY
IsNum	RefName	XY
IsStr		YFromX

File/Directory

CanOpenFile	ExistDir	IsSameFile
ChangeDir	FullPath	ListDirectory
ChooseDir	GetCurDir	MakeDir
ChooseDirectory	GetDriveType	MakeLogicalPath
CloseEchoFile	GetFileInfo	OpenEchoFile
CloseFile	GetFreeDiskSpace	OpenFile

File/Directory

ConvertLocalToUnix	GetLine	ReadFile
ConvertUnixToLocal	GetLongPathName	RelativePath
CopyURL	GetPath	SimetrixFileInfo
CreateLockFile	GetShortPathName	SplitPath
CreateShortcut	GetUncPath	TranslateLogicalPath
DescendDirectories	IsFullPath	

String

Ascii	Parse	SelectRows
Char	ParseParameterString	Sort
Chr	ParseSimplisInit	SortIdx
EscapeString	PathEqual	SprintfNumber
FilterFile	QueryData	StringLength
FormatNumber	ResolveTemplate	StrStr
MakeString	Scan	SubstChar
Mid	Search	SubstString
	SelectColumns	

Model Library

AddModelFiles	GetModelFiles	SearchModels
AssociateModel	IsModelFile	
FindModel	ModelLibsChanged	
GetLibraryModels	RemoveModelFile	

Simulator

GetAnalysisInfo	GetModelParameterValues	GetSimulatorStats
GetAnalysisLines	GetModelType	GetSimulatorStatus
GetConvergenceInfo	GetNonDefaultOptions	GetSoaDefinitions
GetDeviceDefinition	GetPrintValues	GetSoaMaxMinResults
GetDeviceInfo	GetSimplisAbortStatus	GetSoaOverloadResults
GetDeviceParameterNames	GetSimplisExitCode	GetSoaResults
GetDotParamNames	GetSimulationErrors	ParseAnalysis
GetDotParamValue	GetSimulatorMode	SelectAnalysis

Simulator

GetEmbeddedFileName	GetSimulationInfo	SelectSimplisAnalysis
GetInternalDeviceName	GetSimulationSeeds	SimulationHasErrors
GetModelName	GetSimulatorEvents	
GetModelParameterNames	GetSimulatorOption	

Miscellaneous

Date	GetLastError	IsOptionMigrateable
Execute	GetLicenseInfo	IsScript
GetActiveWindow	GetLicenseStats	Progress
GetColours	GetOption	ReadConfigSetting
GetColourSpec	GetPlatformFeatures	ScriptName
GetConfigLoc	GetSimConfigLoc	VersionInfo
GetFileExtensions	GetWindowNames	WriteConfigSetting
GetFonts	HaveFeature	
GetFontSpec	HaveInternalClipboardData	

System

CreateTimer	GetTimerInfo	Shell
DeleteTimer	PutEnvVar	ShellExecute
EditTimer	ReadClipboard	Sleep
GetEnvVar	ReadIniKey	Time
GetPrinterInfo	ReadRegSetting	WriteIniKey
GetSystemInfo	Seconds	WriteRegSetting

Unsupported Functions

A very small number of functions are designated as *unsupported*. These are usually functions we developed for internal use and are not used by the user interface. They are unsupported in so much as we will be unable to fix problems that you may encounter with them.

If you do use an unsupported function and it is useful to you, please tell technical support - by Email preferably. If a number of users find the function useful we will raise its status to supported.

Chapter 4 Function Reference

abs

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real array

Returns absolute value or magnitude of argument. This function is identical to the `mag()` function.

ACSourceDialog

Type	real array
Description	Initial value
Compulsory	Yes
Default	

Return type: Real array

Displays dialog box intended for the user definition of an AC source.

Argument is a real array with two elements which specify the initial values for the two controls as follows:

0	Magnitude
1	Phase

The function returns a real array of length 2 with the same format as the argument described above.

AddConfigCollection

Type	string	string array
Description	Section name	List of entries
Compulsory	Yes	Yes
Default		

Return type: real

Adds a list of entries to a named section in the configuration file.

Argument 1

Section name in configuration file where entries are to be added. The configuration file is where SIMetrix stores its settings. See the *User's Manual* chapter 13 for more information.

Argument 2

List of entries to be added. Note that duplicates are not permitted and any entered will be ignored.

Return Value

The number of new entries successfully added is returned. This will may be less than the number of entries supplied to argument 2 if any are already entered or if their are duplicates in the list supplied.

AddGraphCrossHair

Type	string
Description	Curve Id
Compulsory	Yes
Default	

Return type: string array

Adds a new cursor to the current graph. Note that cursors must be switched on for this to work. This can be done with the command [CursorMode \(page 357\)](#).

For more information on graph annotation objects, please refer to [“Graph Objects” on page 448](#).

Argument 1

Id of curve on which crosshair is initially placed. If the Id supplied is not valid, the cursor will be placed on an undetermined existing curve.

Return value

String array with three elements defined as follows:

Index	Description
0	Id of new cursor
1	Id of cursor's horizontal dimension
2	Id of cursor's vertical dimension

AddModelFiles

Type	string array
Description	List of model files
Compulsory	Yes
Default	

Return type: real

Add files to list of installed models.

Argument is a string array containing the path names of model files to be installed. The names may contain wild cards ('*' or '?') and may also use symbolic constants (see User's Manual). The function does not check the validity of paths supplied.

Return value is the number of files that were successfully installed. A file will only fail to install if it has already been installed.

AddPropertyDialog

Type	string	string	string
Description	initial property name	initial property attribute flags	initial property value
Compulsory	No	No	No
Default	<<empty>>	0	<<empty>>

Return type: string array length 3

Opens the dialog box used to create a new property in the symbol editor. (E.g as opened by Property/Pin/Add Property...)

The first and third arguments initialise the Name and Value boxes respectively. Argument 2 initialises the text location and property attributes. For details on the meaning of attribute flags see [“Attribute flags” on page 404](#).

Return value

String array of length 3 providing the users settings. The function of each element is described below:

0	Property name
1	Flags value
2	Property value

The function returns an empty vector if Cancel is selected.

AddRemoveDialog

Type	string array	string array	string array	string
Description	Initial contents of selected list box	All items available	Options	Box style
Compulsory	Yes	Yes	No	No
Default			<<empty>>	'horizontal'

Return type: string array

Opens a dialog box to allow user to select from a number of items

This dialog box is used by the menu File|Model Library|Add/Remove Models... (horizontal style) and also by the schematic menu View|Configure Toolbar... (vertical style).

The function will display in the lower list box, all items found in both arguments 1 and arguments 2 with no duplicates. In the top list box, only the items found in argument 1 will be displayed. The user may freely move these items between the boxes. The function returns the contents of the top list box as an array of strings.

Argument 3 is a string array of size up to four which may be used to specify a number of options. The first three are used for text messages and the fourth specifies a help topic to be called when the user presses the Help button. The help button will not be shown if the fourth element is empty or omitted.

0	Dialog box caption
1	Label for selected box
2	Label for available box
3	Help topic name

Argument 4 determines the style of the box. The default is 'horizontal' and with this style the two list boxes are on top of each other. If arg4 is set to 'vertical', the two list boxes will be arranged side by side.

The function returns an empty vector if "Cancel" is selected.

AddSymbolFiles

Type	string array
Description	Files to add
Compulsory	Yes
Default	

Return type: real

Adds file or files to list of installed symbol library files.

Argument is a string array containing the path names of the symbol libraries to be installed. The names may use symbolic constants.

arg

Type	real/complex
Description	vector
Compulsory	Yes
Default	

Return type: real

Returns the phase of the argument in degrees. Unlike the functions [phase](#) and [phase_rad](#), this function wraps from 180 to -180 degrees. See [arg_rad](#) function below for a version that returns phase in radians.

arg_rad

Type	real/complex
Description	vector
Compulsory	Yes
Default	

Return type: real

Returns the phase of the argument in radians. Unlike the functions [phase](#) and [phase_rad](#), this function wraps from π to $-\pi$ radians. See [arg](#) function above for a version that returns phase in degrees.

Ascii

Type	string
Description	
Compulsory	Yes
Default	

Return type: real

Returns the ASCII code for the first letter of the argument

AssociateModel

Type	string	string	string	string array
Description	Catalog file (usually OUT.CAT)	User catalog file (usually USER.CAT)	Command to execute to create symbol	Options
Compulsory	Yes	Yes	No	No
Default			<<empty>>	<<empty>>

Return type: real

Special purpose function forms part of parts browser system. Function opens 'Associate Models' dialog box which allows user to associate electrical models with schematic symbols as well as be able to specify part categories and pin mapping. The function modifies the user catalog file (second argument). The return value is FALSE if the user cancels the box otherwise it returns TRUE. For full details on using this dialog box, refer to the "Device Library" chapter in the User's Manual.

The dialog box may be opened in one of two modes namely multiple and single. In multiple mode, a list of models and categories is displayed allowing the association of many devices together. In single mode, a single device name is provided as an argument and only that device may be associated.

To open in single mode, provide a two element string array to argument 4 with the first element set to the model to be associated and the second element set to 'single'. Otherwise the box will be opened in multiple mode in which the first element of argument 4 (if present) defines the initial selected device

atan

Type	real/complex
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Returns the arc tangent of its argument. Result is in radians.

atan_deg

Type	real/complex
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Returns the arc tangent of its argument. Result is in degrees.

BoolSelect

Arguments:			
Type	real	string	string
Description	Initial check box settings	Labels	Dialog Box Caption
Compulsory	No	No	No
Default			

Return type: real array

Opens a dialog box with any number of check boxes. The return value is a real vector containing the user's check box settings. 1 means checked, 0 means not checked. The number of check boxes displayed is the smaller of the length of arguments 1 and 2. If neither argument is supplied, 6 check boxes will be displayed without labels.

If the user cancels the operation, an empty value is returned. This can be checked with the `length()` function.

Example

The following dialog box is displayed after a call to:

```
BoolSelect([0,1,0], ['Label1', 'Label2', 'Label3'], 'Caption')
```



See Also

- [EditSelect page 128](#)
- [RadioSelect page 265](#)
- [ValueDialog page 316](#)

Branch

No arguments

Return Type string

Returns the *branch current formula* for the wire nearest the cursor on the selected schematic. This function will only return a result after the circuit has been netlisted.

The branch current formula is an expression that when evaluated yields the current flowing in the wire. The polarity of the result assumes current flows from right to left and top to bottom. An empty string will be returned if there is more than one path for current to flow or if the wire is dangling.

See Also

- [NearestInst page 240](#)
- [NetName page 241](#)
- [PinName page 255](#)

CanOpenFile

Type	string	string
Description	file name	options
Compulsory	Yes	No
Default		read

Return type: real

Returns TRUE (1) if file specified by argument 1 can be opened otherwise returns FALSE (0). Argument 2 may be set to 'read' (the default) or 'write' specifying what operation is required to be performed on the file.

This function takes account of lock files used to prevent other instances of SIMetrix from opening a file. For example, when a schematic is opened in non read only mode, a lock file is created which will prevent another instance of SIMetrix from opening that file but will not prevent another application from opening the file. CanOpenFile will return false for such files when 'write' mode is specified.

ChangeDir

Type	string
Description	New directory
Compulsory	Yes
Default	

Return type: real

Change current working directory to that specified by argument.

Return value is a code indicating the success of the function:

Code	Meaning
0	Success
1	Cannot create directory
2	Invalid disk. (Windows only)

Char

Type	string	real
Description	Input string	Character position
Compulsory	Yes	Yes
Default		

Return type: string

Returns a string consisting of the single character in `arg1` located at index given in `arg2`. The first character has index 0. An empty string is returned if the index is out of range.

Example

```
Show Char('Hello World!', 4)
```

displays result:

```
Char('Hello World!', 4) = 'o'
```

ChooseDir

Type	string	string	string
Description	Starting directory	Dialog box caption	Message
Compulsory	No	No	No
Default	Current directory	'Choose Directory'	'Double-click directory to select'

Return type:string

Opens a dialog box showing a directory tree. Returns path selected by user or an empty string if cancelled. Initial directory shown specified in argument 1.

ChooseDirectory

Type	string
Description	Starting directory
Compulsory	Yes
Default	

Return type:string

Opens a dialog box showing a directory tree. Returns path selected by user or an empty string if cancelled. Initial directory shown specified in argument1. This function is similar to ChooseDir but uses a different style of dialog box. With Windows it uses the standard system dialog which includes access to network shares. With Linux a design similar to the choose file dialog box is used.

Chr

Type	real
Description	ASCII code
Compulsory	Yes
Default	

Return type: string

Returns a string consisting of a single character specified by an ASCII code. This function may be used to represent special characters such as TAB (Chr(9)) and newline (Chr(10)).

CloseEchoFile

No arguments

Closes the file associated with the Echo command. For more information, see [“OpenEchoFile” on page 246](#).

CloseFile

Type	real
Description	File handle
Compulsory	Yes
Default	

Return type: real

Closes a file opened using [OpenFile \(page 247\)](#).

Argument 1

File handle to close. This is the value returned by the [OpenFile](#) function.

CloseSchematic

Type	real
Description	Schematic handle
Compulsory	Yes
Default	

Return type: real

Closes a schematic handle opened using [OpenSchematic \(page 249\)](#). Schematic handles are used to obtain information about schematics that are not currently being displayed. For more information see “[OpenSchematic](#)” on [page 249](#).

Function returns 1 if successful otherwise returns 0.

CollateVectors

Type	string array	string	real array
Description	Vector names	Group name	Start index, length, division index
Compulsory	Yes	Yes	No
Default			

Return type: real or complex array

Returns the data for the specified vectors in an interleaved manner suitable for writing out in common simulation data formats such as SPICE3 raw format.

Argument 1

List of vector names. Note that they must be valid vector names in the group specified by argument 2. Expressions of vectors are not permitted.

Argument 2

Group name holding vectors specified in argument 1.

Argument 3

Three element array. Element 1 is the start index for the return values, element 2 is the number of values to be returned for each vector and element 3 is the division index. The default values for the three elements are 0, the length of the first vector and 0 respectively.

Return value

If the vectors supplied in arg 1 are real the return value will be a real array. If they are complex the return value will be a complex array. The length of the result will be $3+(\text{number of vectors})\times(\text{vector length})$

The first three elements of the array are:

0: number of vectors
1: start index
2: length of each vector

The remaining elements hold the vector data. This is in the following order:

```
vec1[0]
vec2[0]
vec3[0]
....
vecn[0]
vec1[1]
vec2[1]
vec3[1]
...
vecn[1]
vec1[2]
vec2[2]
vec3[2]
....
vecn[2]
etc.
```

Where vec1 is the first vector specified in arg 1, vec2 the second and so on.

This function is used by the write_raw_file script to create SPICE3 raw file data. The source for this script is provided on the install CD.

CompareSymbols

Type	string	string
Description	symbol name 1	symbol name 2
Compulsory	Yes	Yes
Default		

Return type: real

Returns 1 if the definitions of the schematic symbols specified are identical. Otherwise returns 0. Two symbol definitions are identical if:

1. Their graphics are identical. I.e. all segments, arcs and pin locations are the same
 2. All pin names are the same
 3. All protected properties are identical
- Unprotected properties are not compared.

ComposeDigital

Type	string	real array	string array	string	real
Description	Bus name	Index range	Options	Wire template	Analog thresholds
Compulsory	Yes	No	No	No	No
Default		See notes			[0.8,0.9]

ComposeDigital builds a new vector from a binary weighted combination of digital vectors. It is intended to be used to plot or analyse digital bus signals. The simulator outputs bus signals as individual vectors. To plot a bus signal as a single value - either in numeric or analog form - these individual vectors must be combined as one to create a single value.

Note that ComposeDigital can only process purely digital signals. These are expected to have one of three values namely 0, 1 and 0.5 to represent an invalid or unknown state.

Argument 1

Signal root name. The function expects a range of vectors to be available in a form defined by the *wire template* argument. By default this is in the form *busname#n* where *busname* is specified in argument 1 while the range of values for *n* is specified in argument 2.

Argument 2

Index range. The function processes vectors from *busname#idx_start* to *busname#idx_end*. *idx_start* and *idx_end* are specified by this argument as a two dimensional array. For example if arg 1 is 'BUS' and arg 2 is [0,3], the function will process vectors:

```
BUS1#0
BUS1#1
BUS1#2
BUS1#3
```

as long all 4 vectors exist. If one or more vectors do not exist the first contiguous set of vectors will be used within the indexes specified. So if BUS1#0 didn't exist, the function would use BUS1#1 to BUS1#3. If BUS1#2 didn't exist, it would use just BUS1#0 and BUS1#1.

Note that the index may not be larger than 31.

Argument 3

1 or 2 element string array. Values may be any combination of 'holdInvalid' and 'scale'.

'holdInvalid' determines how invalid states in the input are handled. If the 'holdInvalid' option is specified, they are treated as if they are not present and the previous valid value is used instead. If omitted, invalid states force an output that alternates between -1 or -2. This is to allow consecutive invalid states to be distinguished. For example, suppose there are 4 bits with one bit invalid. If one of the valid bits changes, the end result will still be invalid, but it is sometimes desirable to know that the overall state has changed. So, in this case the first invalid state will show as a -1 and the second invalid state will be -2. In any following invalid state, the result will be -1 and so on.

'scale' forces the output to be scaled by the value $2^{-(idxend - idxstart + 1)}$.

Argument 4

Optional wire template to describe how bus vectors are named. The default is %BUSNAME%#%WIRENUM% which means that bus vectors are of the form *busname#n* where *busname* is the name of the bus (argument 1) and *n* is the index value. For more details about wire templates, see [Netlist \(page 390\)](#).

Argument 5

Threshold used to define logic levels for analog signals. Two element array. The first element is the lower threshold and the second element is the upper threshold. If either or both is omitted these values default to 0.8 and 0.9 respectively.

The lower threshold is the value below which an analog signal is considered to be a logic zero. The upper threshold is the value above which an analog signal is considered to be a logic one.

Return Value

The return value is a real vector that is the binary weighted sum of the vectors defined by arg 1 and arg 2 but treating invalid values (=0.5) as described above. So, in the example above, the result will be:

$$\text{BUS1\#0} + \text{BUS1\#1} \times 2 + \text{BUS1\#2} \times 4 + \text{BUS1\#3} \times 8$$

ConvertLocalToUnix

Type	string
Description	File path
Compulsory	Yes
Default	

Return Type: String

Convert file name to UNIX format using '/' as the directory separator.

Argument 1

Path name.

Return value

On Windows systems, this function returns argument 1 but with any back slash ('\') characters replaced by forward slash ('/'). On Linux systems, argument 1 is returned unchanged.

ConvertUnixToLocal

Type	string
Description	File path
Compulsory	Yes
Default	

Return Type: String

Convert filename to local format using '\' on Windows, '/' on Linux.

Argument 1

Path name

Return value

On Windows systems any '/' found in the input string is replaced by a back slash ('\'). On Linux systems input string is returned unchanged.

CopyURL

Type	string	string	string
Description	From URL file	To URL file	options
Compulsory	Yes	Yes	No
Default			progress

Return Type: String array

Copies a file specified by a URL from one location to another. The URL may specify HTTP addresses (prefix 'http://'), FTP addresses (prefix 'ftp://') and local file system addresses (prefix 'file:').

Argument 1

URL of source file.

Argument 2

URL of destination file

Argument 3

Options: can be 'progress' or 'noprogess'. If set to 'progress' (the default) a box will display with a bar showing the progress of the file transfer. Otherwise no such box will display.

Return Value

String array of length 2. First element will be one of the values shown in the following table:

Id	Description
UserAbort	User aborted operation
TimedOut	Connection timed out. This error usually occurs when an Internet connection is down.
NoError	Operation completed successfully
Unexpected1	This is an internal error that should not occur
UnknownProtocol	The protocol is unknown. I.e the URL prefix is not implemented. (Only HTTP, FTP and FILE are implemented)
Unsupported	This is an internal error that should not occur
ParseError	The URL does not comply with the expected format

Id	Description
IncorrectLogin	A username and password are required for this URL
HostNotFound	The specified host in the URL could not be found. This error can also occur if there is no Internet connection.
Unexpected2	This is an internal error that should not occur
MkdirError	Could not create target directory
RemoveError	This is an internal error that should not occur
RenameError	This is an internal error that should not occur
GetError	An error occurred while fetching a file
PutError	An error occurred while storing a file
FileNotExist	File doesn't exist
PermissionDenied	You do not have sufficient privilege to perform the operation
Unknown Error	This is an internal error that should not occur

The second element of the returned string gives a descriptive message providing more information about the cause of failure.

COS

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Return cosine of argument. Result is in radians.

cos_deg

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Return cosine of argument. Result is in degrees.

CreateDiodeDialog

Type	string array
Description	Initial values
Compulsory	No
Default	

Return type: string array

Opens a specialised dialog used by the diode model in circuit parameter extractor. See internal script `make_srdiode_model` for an application example of this function.

Argument 1

String array providing initial values for the various controls. The order is 'IF', 'IRM', 'df/dt', 'Tr', 'Vd1', 'Id1', 'Vd2', 'Id2', 'Cj0', 'Save option', 'Device name'. The 'Save option' will be '0' if 'Save to schematic symbol' is specified and '1' if 'Save to model library' is specified.

Return value

String array corresponding exactly to argument 1 and holding the user's selected values. Return value will be empty if the user cancels the box.

CreateLockFile

Type	string	string
Description	filename	operation
Compulsory	Yes	Yes
Default		

Return Type: string

Creates or removes a lock file for the filename specified. This can be used to synchronise operations between multiple instances of SIMetrix.

Argument 1

Filename to lock. The lock file created will have the same name with the suffix `.lck`. On Windows the file will be locked for write and other applications will not be able to delete or write to the file. On Linux the file will have a cooperative lock applied.

Argument 2

One or two element string array. First element is the operation to be performed. This is either 'lock' or 'unlock'. If 'lock' is specified, an attempt will be made to create a lock file. The operation will fail if the file has already been locked - perhaps by another instance of SIMetrix. If 'unlock' is specified the file will be removed provided that this instance of SIMetrix created the file in the first place.

A second element may be specified and set to 'autodelete'. In this case the file will automatically be unlocked when control is returned to the command line.

Return value

May be one of the following values:

success	Operation successful
failed	Lock failed because the file has already been locked
notexist	Attempt made to unlock a file that was not locked by this instance or has not been locked at all
locked	File has already been locked by this instance

CreateShortcut

Type	string	string	string
Description	Path of object	Path of link file	Description
Compulsory	Yes	Yes	Yes
Default			

Return type: string

This function is only available on the Windows platform.

Create a 'shortcut' to a file or directory.

Argument 1

Path of file or directory which shortcut will point to

Argument 2

Path of shortcut itself.

Argument 3

Description of shortcut

Return Value

‘Success’ or ‘Fail’

CreateTimer

Type	string	real	string
Description	Command	Interval	options
Compulsory	Yes	Yes	No
Default			2

Return type: real

Creates a timer to run a script at regular intervals or at some specified time in the future.

Argument 1

Command to run. This can be a primitive command or the name of a script and may include arguments to the command or script.

Argument 2

Interval in milliseconds. The first event will occur after the interval time has elapsed.

Argument 3

Options. May be set to ‘oneshot’ in which case the timer event is triggered just once.

Return Value

The function returns an integer id. This can be used as an argument to functions [DeleteTimer](#) (page 110), [EditTimer](#) (page 128) and [GetTimerInfo](#) (page 203)

CyclePeriod

Type	real vector	real	real	real
Description	Input vector	Baseline	Interpolation order	X start position (0, 1 or 2)
Compulsory	Yes	Yes	No	No
Default			2	0

Return type:

Returns the time between zero crossing pairs with the same slope direction. It can be used for plotting frequency vs time by using $1/\text{CyclePeriod}$.

Argument 1

Input vector to be processed.

Argument 2

Baseline for zero-crossing detection.

Argument 3

Interpolation order, may be 1 or 2. The actual zero crossing point from which the measurements are based are calculated by interpolation from points either side of the zero-crossing. This sets the order of the interpolation algorithm

Argument 4

Can be 0, 1 or 2. This shifts the x-axis of the result. So for example if the input vector is a 1kHz sine wave, the first element of the result will be the duration of the first cycle - i.e 1mS. What this argument does is set what the x value will be. If set to 0, it will be 1mS - i.e the location of the end of the first cycle. If set to 1, it will be 0.5mS - i.e the location of the end of the first half-cycle and if set to 2, it will be 0, i.e the start of the input.

CV

This function behaves identically to [GetCurveVector \(page 153\)](#)

Date

Type	string
Description	format
Compulsory	No
Default	'locale'

Return type: string

Returns the current date in the format specified.

Argument 1

May be 'iso' or 'locale'. When set to 'locale' the date is returned in a format specified by system settings. When set to 'iso' the date is returned in a format complying with ISO8601 which is YYYY-MM-DD where YYYY is the year, MM is the month of the year (between 01 and 12), and DD is the day of the month between 01 and 31.

db

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Returns $20 * \log_{10}(\text{mag}(\text{argument}))$

DCSourceDialog

Type	real
Description	Initial value
Compulsory	Yes
Default	

Return type: real

Opens “Edit DC Source” dialog box. This accepts user input for the value of a DC source.

Return value is the user's entry

DefineADCDialog

Type	real array
Description	initial values
Compulsory	Yes
Default	

Return type: real array

Opens a dialog box to define an analog to digital converter.

Argument is a real array which specifies the initial values for each control as follows:

Element index	Description
0	Number of bits
1	Convert time (default 1u)
2	Maximum conversion rate (default 2Meg)
3	Offset voltage (default 0)
4	Range (default 5)

The function returns a real array of length 5 with the same format as the argument described above. If the user selects “Cancel” the function returns an empty vector.

DefineArbSourceDialog

Type	string array
Description	initial values
Compulsory	Yes
Default	

Return type: string array

Opens a dialog box to define an arbitrary source:

Argument is a string array which specifies the initial values for each control as follows:

Element index	Description
0	Expression
1	Number of input voltages. (Default 1. Must be entered as a string)
2	Number of input currents. (Default 0. Must be entered as a string)
3	Output config. 0: Single ended voltage (default) 1: Single ended current 2: Differential voltage 3: Differential current (value must be entered as a string)

The function returns a string array of length 4 with the same format as the argument described above.

DefineBusPlotDialog

Type	string array	string
Description	Initial values	options
Compulsory	Yes	No
Default		

Return type: string array

Opens a dialog box to allow the user to plot a bus.

Argument 1

String array of length up to 9. Elements defined in the following table:

Index	Description	Default
0	Bus name	''
1	Bus start index	0
2	Bus end index	0
3	Display type: '0' Decimal '1' Hexadecimal '2' Analog waveform '3' Binary	'0'
4	Hold invalid - default 'TRUE' Hold invalid ON 'FALSE' Hold invalid off	'FALSE'
5	Scale factor	'1.0'
6	Offset	'0.0'
7	Units	
8	Items used to load 'Units' combo box separated by ' '	
9	Analog threshold lower	
10	Analog threshold upper	

Index	Description	Default
11	Axis type: 0: Auto select 1: Use separate Y-axis 2: Use separate grid 3: Digital	
12	Axis name	
13	Use separate graph? 0, yes; 1, no	
14	Graph name	

Argument 2

Options. Currently just one. If set to 'noProbeOptions', the Probe Options sheet will be hidden.

Return Value

String array with the same length as the input. Each field holds the value selected by the user. Note that field index 8 does not currently output a meaningful value and should be ignored.

DefineCounterDialog

Type	real array
Description	initial values
Compulsory	Yes
Default	

Return type: real array

Opens a dialog box to define a digital counter.

Argument is a real array which specifies the initial values for each control as follows:

0	Number of bits
1	Maximum count (default = $2^{\text{number of bits}-1}$)
2	1 = Has reset, 0 = does not have reset (default 0)
3	Clock to out delay (default 10n)

The function returns a real array of length 4 with the same format as the argument described above. If the user selects "Cancel" the function returns an empty vector.

DefineCurveDialog

Type	string array
Description	initial values
Compulsory	Yes
Default	

Return type: string array

Opens the dialog box used to define a curve for plotting. See menu Graphs and Data|Add Curve... .

The argument is a string array of length 25 which defines how the various controls are initialised. This array has the same format for EditAxisDialog ([page 113](#)) and EditProbeDialog ([page 123](#)). Not all the elements are relevant to this function. The following table describes the elements that are used.

Index	Purpose	Notes	Default
0	Axis Type	Setting of Axis Type group in Axis/Graph Options sheet. Possible values: 'auto' Auto Select 'selected' Use Selected 'axis' Use New Y-Axis 'grid' Use New Grid 'digital' Digital	No default. Value must be specified.
1	Graph Type	Setting of Graph Options group in Axis/Graph Options sheet. Possible values: 'add' Add To Selected 'newsheet' New Graph Sheet 'newwindow' New Graph Window	No default. Value must be specified.
2	Axis name	Not used with this function	
3	Persistence	Not used with this function	
4	Graph name	Not used with this function	
5	Curve label	Curve label control in Define Curve sheet	<<empty>>
6	Analysis	Not used with this function	
7	Plot on completion	Not used with this function	
8	reserved for future use	Not used with this function	

Index	Purpose	Notes	Default
9	reserved for future use	Not used with this function	
10	X axis graduation	Setting of Log Lin Auto for X Axis in Axis Scales sheet. Possible values: 'in' Lin 'log' Log 'auto' Auto	'auto'
11	X axis scale options	Setting of scale options for X Axis in Axis Scales sheet. Possible values: 'nochange' No Change 'auto' Auto scale 'defined' Defined	'auto'
12	Y axis graduation	Setting of Log Lin Auto for Y Axis in Axis Scales sheet. Possible values as for X axis.	'auto'
13	Y axis scale options	Setting of scale options for X Axis in Axis Scales sheet. Possible values as for X axis.	'auto'
14	X axis min limit	Min value for X Axis in Axis Scales sheet. Must be specified as a string.	0
15	X axis max limit	Max value for X Axis in Axis Scales sheet. Must be specified as a string.	1
16	Y axis min limit	Min value for Y Axis in Axis Scales sheet. Must be specified as a string.	0
17	Y axis max limit	Max value for Y Axis in Axis Scales sheet. Must be specified as a string.	1
18	X axis label	Axis Label setting for X-Axis group in Axis Labels sheet	<<empty>>
19	X axis units	Axis Units setting for X-Axis group in Axis Labels sheet	<<empty>>
20	Y axis label	Axis Label setting for Y-Axis group in Axis Labels sheet	<<empty>>

Index	Purpose	Notes	Default
21	Y axis units	Axis Units setting for Y-Axis group in Axis Labels sheet	<<empty>>
22	Y-expression	Contents of Y expression edit box	<<empty>>
23	X-expression	Contents of X expression edit box, if enabled	<<empty>>
24	Vector filter	Subcircuit filter selection in Available Vectors group. Possible values 'all' All 'top' Top level sub circuit name Select a subcircuit name.	

The available vectors list box is initialised with the names of vectors in the current group.

The function returns a string array with the same format as the argument. If the user selects Cancel the function returns an empty vector.

DefineDACDialog

Type	real array
Description	initial values
Compulsory	Yes
Default	

Return type: real array

Opens a dialog box to define an analog to digital converter.

Argument is a real array which specifies the initial values for each control as follows:

0	Number of bits
1	Output slew time (10n)
2	Offset voltage (default 0)
3	Range (default 5)

The function returns a real array of length 4 with the same format as the argument described above. If the user selects "Cancel" the function returns an empty vector.

DefineFourierDialog

Type	string array	real array
Description	initial values	sample vector
Compulsory	Yes	No
Default		

Return type: string array

Opens the Define Fourier dialog box used to specify a fourier transform. This is similar to the Define Curve dialog (see [page 99](#)) but has an extra tabbed sheet to define the fourier analysis options. Select menu Graphs and Data\Fourier... to see how this dialog box looks.

The function takes an argument that is a string array with up to 37 elements which initialises the controls in the dialog box. The first 25 have the same function as for the DefineCurveDialog() function (see [page 99](#)). The remaining are described in the following table:

Index	Description	Default
0-24	See " DefineCurveDialog " on page 99	
25	Fundamental Frequency	100
26	Frequency display - Start Frequency	<<empty>>
27	Frequency display - Stop Frequency	10K
28	Number of points for FFT interpolation	256 if arg 2 not specified. See below
29	Interpolation order for FFT	2
30	Fourier method. Possible values: 'continuous' Use continuous fourier 'interpolated' Use interpolated FFT	'continuous'
31	Window function. Possible values: 'rectangular' 'hanning' 'hamming' 'blackman'	'hanning'
32	Start of data span	0
33	End of data span	0.01

Index	Description	Default
34	Use specified span: TRUE/FALSE	FALSE
35	Know fundamental frequency: TRUE/FALSE	FALSE
36	Resolution	100
37	Plot options - 'mag', 'db' or 'phase'	

A second argument may be specified to provide time domain information. Usually this would be the 'time' vector created by the simulation. The vector is analysed to find the start time, stop time and number of interpolation points. The number of interpolation points is calculated from the number of points in the time vector and is the next highest integral power of 2.

The function returns a string array with the same format as the argument. If the user selects Cancel the function returns an empty vector.

DefineIdealTxDialog

Type	real array	real array	real array	string
Description	initial inductance, coupling factors, number of windings	initial primary turns ratios	initial secondary turns ratios	options
Compulsory	Yes	Yes	Yes	No
Default				

Return type:real array

Opens a dialog box to define an ideal transformer.

Argument 2

Real array of size 6. Function of each element is described below:

0	Primary 1 inductance
1	Coupling factor primary to primary
2	Coupling factor secondary to secondary
3	Coupling factor primary to secondary
4	Number of primaries
5	Number of secondaries

Argument 2

Real array of primary turns ratios relative to primary 1. (The first value is the ratio of primary 1 to itself. This is of course always 1 but the value is in fact ignored).

Argument 3

Real array of secondary turns ratios relative to primary 1.

Argument 4

If set to 'nonind', the box design will that used for non-inductive transformers. These do not show inductance related parameters.

Return value

The function returns, the settings selected by the user in a single real array with the same format as the three arguments concatenated together. If the user selects Cancel the function returns an empty vector.

DefineLaplaceDialog

Type	string array
Description	initial settings
Compulsory	Yes
Default	

Return type: string array

Opens a dialog box to define a Laplace transfer function.

The argument is a string array of length 5 that defines the initial settings. The meaning of each element is as follows:

Element index	Description
0	Laplace expression. (contents of “Define output using s variable” box)
1	Device type: 0 Transfer function 1 Impedance - V/I 2 Admittance - I/V
2	Input type: 0 Single ended voltage 1 Single ended current 2 Differential voltage 3 Differential current
3	Output type 0 Single ended voltage 1 Single ended current 2 Differential voltage 3 Differential current
4	Frequency scale factor

The function returns a string array of length 5 with the same format as the argument described above. If the user selects “Cancel” the function returns an empty vector.

DefineLogicGateDialog

Type	real array
Description	initial settings
Compulsory	Yes
Default	

Return type: real array

Opens a dialog box to define a logic gate.

The argument is a real array of length 3 and defines the initial settings for the box controls as follows:

Index	Description
0	Number of inputs
1	Propagation delay
2	Gate type:
0	AND
1	NAND
2	OR
3	NOR

The function returns a real array of length 3 with the same format as the argument described above. If the user selects Cancel the function returns an empty vector.

DefinePerfAnalysisDialog

Type	string array
Description	Initial values
Compulsory	Yes
Default	

Return type: string array

Essentially the same as DefineCurveDialog but with a different design for the expression entry. Used by the Probe|Performance Analysis... and Probe|Plot Histogram... menus.

DefineRegisterDialog

Type	real array
Description	initial settings
Compulsory	Yes
Default	

Return type: real array

Opens a dialog box to define a bus register.

The argument is a real array of length 4 and defines the initial settings for the box controls as follows:

0	Number of bits
1	1 if "Has output enable" box checked. Otherwise 0.
2	Setup time
3	Clock delay

The function returns a real array of length 4 with the same format as the argument described above. If the user selects Cancel the function returns an empty vector.

DefineRipperDialog

Type	string array	string array
Description		
Compulsory	Yes	Yes
Default		

Return type:string array

Opens a dialog box to define a schematic bus ripper.

Argument 1

This argument is a string array of length 4 and defines the initial settings for the box controls as follows:

Index	Description
0	Bus name
1	Start index (entered as a string)
2	End index (entered as a string)
3	Style index. This is an index into the values in the style box which are defined in argument 2

Argument 2

String array containing list of items entered in style box

Return value

The function returns a string array of length 4 with the same format as argument 1 described above. If the user selects Cancel the function returns an empty vector.

DefineSaturableTxDialog

Type	string array	string array	real array	real array
Description	Core material info	Part number info	Winding ratios	Initial values
Compulsory	Yes	Yes	Yes	Yes
Default				

Return type: real array

Opens a dialog box to define a saturable transformer.

Argument 1

Array of core material specifications. Each element is a string has the format:
name;model_name;saturation_flux_density

Argument 2

Array of core part specifications. Each element is a string which has the format:
name;Ae;Le;Ue;material_name

Argument 3

Array of turns ratios.

Argument 4

Real array with up to 9 elements that defines the initial values for the controls in the dialog box, as defined in the following table

Index	Description
0	Primary number of turns
1	Selected material index (into arg 1)
2	Selected part index (into arg 2). -1 for manual entry.
3	Number of primaries
4	Number of secondaries
5	Effective area
6	Effective length
7	Ue
8	Coupling factor

Return Value

The return value is a real array containing the user's selection. The definition of the values is identical to that for argument 4 as described above.

DefineSimplisMultiStepDialog

Type	string array	string array
Description	Configuration	
Compulsory	Yes	Yes
Default		

Return type: string array

Opens a dialog box used to define SIMPLIS multi step analyses.

Argument 1

4 element string array used to initialise the dialog box as defined by the following table:

Index	Description
0	Sweep mode: 'Parameter' or 'MonteCarlo'
1	Parameter name
2	Step type: 'Decade', 'Linear' or 'List'
3	Group curves (true/false)

Argument 2

Sweep values. If step type is decade or linear, values define start, stop and number of steps. Otherwise defines list of values.

DefineShiftRegDialog

Type	real array
Description	
Compulsory	Yes
Default	

Return type: real array

Open a dialog box to define a shift register.

The argument is a real array of length 2 and defines the initial settings of the box controls as follows:

Index	Description
0	Number of inputs
1	Clock to out delay

The function returns a real array of length 3 with the same format as the argument described above. If the user selects Cancel the function returns an empty vector.

DeleteConfigCollection

Type	string
Description	Section name
Compulsory	Yes
Default	

Return type: real

Deletes an entire section in the configuration file.

Argument 1

Name of section to be deleted.

Return Value

Returns the number of entries successfully deleted.

DeleteTimer

Type	real
Description	Timer ID
Compulsory	Yes
Default	

Return type: real

Deletes a timer

Argument 1

Timer ID as returned by [CreateTimer](#) (page 93)

Return Value

Returns 1.0 if the function is successful, otherwise returns 0.0. The function will fail if the timer specified does not exist.

DescendDirectories

Type	string
Description	Directory
Compulsory	Yes
Default	

Return type: string array

Returns all directories under the specified directory. DescendDirectories recurses through all sub-directories including those pointed to by symbolic links. DescendDirectories only returns directory names. It does not return files. Use the ListDirectory function to return the files in a directory.

DescendHierarchy

Type	string	real
Description	Property used to report 'where used' references	Schematic handle
Compulsory	No	No
Default	'Ref'	

Return type: string array

Descends through the hierarchy from the current schematic and collects each distinct schematic in use. The result is a list of schematic path names. Each path name is accompanied by a list of hierarchy references where that schematic is used.

Argument 1

Name of property to be used to report 'where used' references. Each entry in the return value contains a list of schematic instance references that identify where the schematic component is used. The references are in the form of a series of property values separated by a period ('.'). The property used defaults to 'Ref' but this argument may be used to identify another property - e.g. 'Handle'.

Argument 2

Schematic handle as returned by the OpenSchematic function (page 249). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

Return Value

Returns a string array with one element for each schematic file used in the hierarchy. Each element is a semi-colon delimited list of values. The first value is the full path to the schematic in UNC form if applicable. On Windows UNC paths begin with '\\' followed by a server name and path. Paths referenced by a local drive letter are not returned in UNC form even if sharing is enabled for that drive. UNC is not supported on Linux and instead all paths will be returned in canonical form with all links resolved.

The remaining values are a list of hierarchical references identifying where that schematic is used within the hierarchy. The references use the value of the property defined in argument 1.

diff

Type	real array
Description	vector
Compulsory	Yes
Default	

Return type: real array

Returns the derivative of the argument with respect to its reference. If the argument has no reference the function returns the derivative with respect to the argument's index - in effect a vector containing the difference between successive values in the argument. For details on references see "[Vector References](#)" on page 42.

EditArcDialog

Type	real	real
Description	initial start to finish angle	initial ellipse height/width
Compulsory	No	No
Default	90	1

Return type: real array

Opens a dialog box used to define an arc circle or ellipse for the symbol editor.

The function takes two arguments that define initial values for the start to finish angle and the ellipse height/width.

The function returns a real array of length 2 that contain the user settings as follows:

0	Start to finish angle
1	Ellipse height/width

If the user selects Cancel the function returns an empty vector.

EditAxisDialog

Type	string array
Description	initial settings
Compulsory	Yes
Default	

Return type: string array

Opens a dialog box used to edit graph axes

The argument is a string array of length 25 which defines how the various controls are initialised. This array has the same format as DefineCurveDialog ([page 99](#)) and EditProbeDialog ([page 123](#)) but not all the elements are used here. The following table describes the elements that are used.

Index	Purpose	Notes	Default
0-10		Not used with this function	
11	X axis scale options	Setting of scale options for X Axis in Axis Scales sheet. Possible values: 'nochange' No Change 'auto' Auto scale 'defined' Defined	'auto'
12	Y axis graduation	Not used with this function	
13	Y axis scale options	Setting of scale options for X Axis in Axis Scales sheet. Possible values as for X axis.	'auto'
14	X axis min limit	Min value for X Axis in Axis Scales sheet. Must be specified as a string.	0

Index	Purpose	Notes	Default
15	X axis max limit	Max value for X Axis in Axis Scales sheet. Must be specified as a string.	1
16	Y axis min limit	Min value for Y Axis in Axis Scales sheet. Must be specified as a string.	0
17	Y axis max limit	Max value for Y Axis in Axis Scales sheet. Must be specified as a string.	1
18	X axis label	Axis Label setting for X-Axis group in Axis Labels sheet	<<empty>>
19	X axis units	Axis Units setting for X-Axis group in Axis Labels sheet	<<empty>>
20	Y axis label	Axis Label setting for Y-Axis group in Axis Labels sheet	<<empty>>
21	Y axis units	Axis Units setting for Y-Axis group in Axis Labels sheet	<<empty>>
22	Y-expression	Not used with this function	
23	X-expression	Not used with this function	
24	Vector filter	Not used with this function	

The function returns a string array with the same format as the argument. If the user selects Cancel the function returns an empty vector.

EditBodePlotProbeDialog

Type	string array
Description	Initialisation
Compulsory	No
Default	

Return Type: string array

UI function for editing Bode plot fixed probes.

Argument 1

Array values used to initialise dialog as shown in the table below.

Index Description

0	Gain label
1	Phase label
2	Persistence
3	'Multiplied by -1' . '0' for normal, '1' for invert
4	'Use dB auto limits'. '1' on, '0' off
5	Minimum limit - dB
6	Maximum limit - dB
7	'Use phase auto limits'. '1' on, '0' off
8	Minimum limit - phase
9	Maximum limit - phase

Return Value

Returns the values entered in the dialog controls as defined in the table above

EditCrosshairDimensionDialog

Type	string array	string array	string array
Description	Property names	Property values	Property types
Compulsory	Yes	Yes	No
Default			

Return type:string array

Opens a dialog intended for editing the characteristics of cursor crosshair dimensions.

The Properties sheet behaves in the same way as the EditObjectPropertiesDialog function (see [page 120](#)) and is initialised by the function's arguments. The Edit sheet allows the edit and display of certain properties as defined in the following table:

Property Name Affects Control:

Label1	Label 1
--------	---------

Property Name	Affects Control:
Label2	Label 2
Label3	Label 3
Style	Contents of Style box. One of six values: Auto: Automatic, Show Difference Internal Internal, Show Difference External External, Show Difference P2P1 Show Absolute P2P1AutoAutomatic, Show Difference, Show Absolute None No controls selected
Font	Font. String defining font specification

If any of the controls in the **Edit** sheet are changed, the corresponding property values in the **Properties** sheet will reflect those changes and vice-versa.

EditCurveMarkerDialog

Type	string array	string array	string array
Description	Property names	Property values	Property types
Compulsory	Yes	Yes	No
Default			

Return type:string array

Opens a dialog intended for editing the characteristics of curve markers.

The Properties sheet behaves in the same way as the EditObjectPropertiesDialog function (see [page 120](#)) and is initialised by the functions arguments. The Edit sheet allows the edit and display of certain properties as defined in the following table:

Property Name	Affects Control:
Label	Label
LabelJustification	Text Alignment box. One of these values: -1 Automatic 0 Left-Bottom 1 Centre-Bottom 2 Right-Bottom 3 Left-Middle 4 Centre-Middle 5 Right-Middle 6 Left-Top 7 Centre-Top 8 Right-Top
Font	Font. String defining font specification

If any of the controls in the Edit sheet are changed, the corresponding property values in the Properties sheet will reflect those changes and vice-versa.

EditDeviceDialog

Type	string array	string array	string array	string array
Description	options/initial settings	devices	parameter names	parameter values
Compulsory	Yes	Yes	No	No
Default			<<empty>>	<<empty>>

Return type:string array

Opens a dialog box used to select a device and optionally specify its parameters.

Argument 1

Defines options and initial settings as follows:

Index	Description
0	Text entered in edit control above list box. If the text item is also present in the device list (argument 2), then that item will be selected.
1	Ignored unless element 1 is empty. Integer (entered in string form) which defines selected device.
2	Dialog box caption. Default if omitted: "Select Device"
3	Message at the top of the dialog box. . Default if omitted: "Select Device"

Argument 2

String array defining the list of devices.

Argument 3

String array defining list of parameter names. See argument 4.

Argument 4

String array defining list of parameter values. If arguments 3 and 4 are supplied the "Parameters..." button will be visible. This button opens another dialog box that provides the facility to edit these parameters' values.

Return value

The function returns a string array in the following form

Index	Description
0	Entry in the text edit box.
1	Index into device list (argument 2) of device in text edit box. If this device is not in the list, -1 will be returned.
2	Number of parameter values.
3 onwards	The values of the parameters in the order they were passed.

If the user selects Cancel the function returns an empty vector.

EditDigInitDialog

Type	real array
Description	initial setting
Compulsory	Yes
Default	

Return type: real array

Opens a dialog box used to define a digital initial condition

The argument is a real array of length 2 which defines the initial settings of the dialog box as follows:

1	Initial state:		
	1	ONE	
	0	ZERO	
2	Initial Strength		
	1	Strong	
	0	Resistive	

The function returns a real array of length 2 with the same format as argument 1 described above. If the user selects “Cancel” the function returns an empty vector.

EditFreeTextDialog

Type	string array	string array	string array
Description	Property names	Property values	Property types
Compulsory	Yes	Yes	No
Default			

Return type: string array

This function is almost identical to the EditCurveMarkerDialog functions except for some changes to the aesthetics of the dialog box.

EditGraphTextBoxDialog

Type	string array	string array	string array
Description	Property names	Property values	Property types
Compulsory	Yes	Yes	No
Default			

Return type: string array

Opens a dialog intended for editing the characteristics of text box objects for graphs.

The Properties sheet behaves in the same way as the “[EditObjectPropertiesDialog](#)” (see [page 120](#)) and is initialised by the function’s arguments. The Edit sheet shown above allows the edit and display of certain properties as defined in the following table:

Property Name	Affects Control:
Label	Label
Colour	Background Colour. An integer defining the RGB value
Font	Font. String defining font specification

If any of the controls in the Edit sheet are changed, the corresponding property values in the Properties sheet will reflect those changes and vice-versa.

EditLegendBoxDialog

Type	string array	string array	string array
Description	Property names	Property values	Property types
Compulsory	Yes	Yes	No
Default			

Return type: string array

This function is virtually identical to “[EditGraphTextBoxDialog](#)” above except for a different caption.

EditObjectPropertiesDialog

Type	string array	string array	string array	string array
Description	Property names	Property values	Property types	Options
Compulsory	Yes	Yes	No	No
Default				

Return type:string array

Displays a dialog box allowing the editing of property values. This is used for a number of functions. See the schematic right-click popup menu Edit Properties... for an example.

Arguments 1 and 2

Function will list in a dialog box the property names and values given in the first two arguments. The function returns the values of the properties. Unless declared read-only (see below) the value of each property may be edited by the user by double clicking on its entry in the list.

Argument 3

The third argument of the function declares the type for each property. Possible values are:

'String'	Property value is a simple text string
'Font'	Property value is a font definition. When the user double clicks the item to edit, a font dialog box will be opened. Font definitions consist of a series of numeric a text values separated by semi-colons. E.g. '-11;0;0;0;0;Arial'
'Colour'	Property value is a colour definition. When the user double clicks the item to edit, a "choose colour" dialog box will be opened. Colours are defined by a single integer that specifies the colour's RGB value.
item1 item2 item3 ...	Up to six items separated by the ' ' symbol. When the user double clicks a property so defined, a dialog showing a number of radio buttons is displayed labelled <i>item1</i> , <i>item2</i> etc. The value of the property will be the item selected.
'*'	Declares the property read-only. If the user attempts to edit its value a warning message box will be displayed.

Argument 4

Array of up to 4 values as described in the following table:

Index	Description	Default
0	Box caption	'Edit Properties'
1	Properties tabbed sheet tab title	'Properties'
2	Name column title	'Name'
3	Value column title	'Value'

Note that fields 2 and 3 should be provided as a pair. If 2 is supplied but not 3, 2 will be ignored and the default value will be used.

Return Value

String array containing values for all properties.
An empty result is returned if the user cancels the dialog box.

EditPinDialog

Type	string	string
Description	initial pin name	initial flags value
Compulsory	Yes	No
Default		'256'

Return type: string array

Opens a dialog box used to edit a pin in the symbol editor.

The first argument specifies the initial value for the Pin name entry. The second argument specifies the initial value for the remaining controls using the property attributes flag. See [“Attribute flags” on page 404](#) for details.

The function returns a string array of length 2 with the following definition:

0	Flags value defining justification and property attributes.
1	Pin name

If the user selects Cancel the function returns an empty vector.

EditPotDialog

Type	real array
Description	initial settings
Compulsory	Yes
Default	

Return type: real array

Opens a dialog to define a potentiometer

The argument is a real array of length 3 and defines the initial settings as follows:

0	Resistance
1	Wiper position (0 to 1)
2	Run simulation after position changed check box state
	1 checked
	0 not checked

The function returns a string array with the same format as the argument. If the user selects “Cancel” the function returns an empty vector.

EditProbeDialog

Type	string array
Description	initial settings
Compulsory	Yes
Default	

Return type: string array

Opens a dialog to define a schematic fixed probe

The argument is a string array of length 25 which defines how the various controls are initialised. This array has the same format for EditAxisDialog ([page 113](#)) and DefineCurveDialog ([page 99](#)). Not all the elements are relevant to this function. The following table describes the elements that are used.

Index	Purpose	Notes	Default
0	Axis Type	Setting of Axis Type group in Probe Options sheet. Possible values: 'auto' Auto Select 'axis' Use New Y-Axis 'grid' Use New Grid 'digital' Digital	No default. Value must be specified.
1	Graph Type	Not used with this function	
2	Axis name	Entry in Axis Name in Probe Options sheet	<<empty>>
3	Persistence	Entry in Persistence box in Probe Options sheet	<<empty>>
4	Graph name	Entry in Graph Name in Probe Options sheet	<<empty>>
5	Curve label	Curve label control in Probe Options sheet	<<empty>>

Index	Purpose	Notes	Default
6	Analysis	Setting for Analyses check boxes in "Probe Options" sheet. Single string comprising a combination of "ac", "dc" and "tran" separated by the pipe symbol (' '). An empty string will cause all boxes to be checked and "none" will clear all boxes.	<<empty>>
7	Plot on completion	State of Plot on completion only check box. 'true' Checked 'false' Not checked	'false'
8	reserved for future use	Not used with this function	
9	reserved for future use	Not used with this function	
10	X axis graduation	Setting of Log Lin Auto for X Axis in Axis Scales sheet. Possible values: 'lin' Lin 'log' Log 'auto' Auto	'auto'
11	X axis scale options	Setting of scale options for X Axis in Axis Scales sheet. Possible values: 'nochange' No Change 'defined' Defined	'auto'
12	Y axis graduation	Setting of Log Lin Auto for Y Axis in Axis Scales sheet. Possible values as for X axis.	'auto'
13	Y axis scale options	Setting of scale options for X Axis in Axis Scales sheet. Possible values as for X axis.	'auto'
14	X axis min limit	Min value for X Axis in Axis Scales sheet. Must be specified as a string.	0
15	X axis max limit	Max value for X Axis in Axis Scales sheet. Must be specified as a string.	1
16	Y axis min limit	Min value for Y Axis in Axis Scales sheet. Must be specified as a string.	0
17	Y axis max limit	Max value for Y Axis in Axis Scales sheet. Must be specified as a string.	1
18	X axis label	Axis Label setting for X-Axis group in Axis Labels sheet	<<empty>>

Index	Purpose	Notes	Default
19	X axis units	Axis Units setting for X-Axis group in Axis Labels sheet	<<empty>>
20	Y axis label	Axis Label setting for Y-Axis group in Axis Labels sheet	<<empty>>
21	Y axis units	Axis Units setting for Y-Axis group in Axis Labels sheet	<<empty>>
22	Y-expression	Not used with this function	
23	X-expression	Not used with this function	
24	Vector filter	Not used with this function	
25	Curve colour	Colour of curve as an RGB value. May be passed directly to the .GRAPH colour parameter	<<empty>>

The function returns a string array with the same format as the argument. If the user selects Cancel the function returns an empty vector.

EditPropertyDialog

Type	string	string	string	string
Description	property name	initial property flags	initial property value	option
Compulsory	Yes	No	No	No
Default	<<empty>>	0	<<empty>>	

Return type: string array

Opens a dialog box intended to edit a property in both the symbol and schematic editors. Select the symbol editor's Property/Pin|Edit Property... menu then double click on one of the items. This will open this dialog box.

The first argument specifies the property name and this is displayed at the top left of the box. This cannot be edited by the user. The third argument initialises the Value box while argument 2 initialises the text location and property attributes using the property flag value. For details on the meaning of flags values see "[Attribute flags](#)" on page 404 for details.

Return value

String array of length 2 providing the users settings. The function of each element is described below:

0	Flags value
1	Property value

The function returns an empty vector if Cancel is selected.

EditReactiveDialog

Type	string array	string	string	string
Description	Initial values	Options	Parameter names	Parameter values
Compulsory	Yes	Yes	No	No
Default				

Return type: string array

Opens a dialog box designed to edit inductors and capacitors. Arguments have the following meaning:

Argument 1

First element is the initial value of device. Second element is the initial condition.

Argument 2

Three element string array. Each field has the meaning defined in the following table:

Index	Description
0	Caption for value group box
1	Initial range. Possible values: 'E6', 'E12', 'E24'
2	Type of device. Possible values: 'capacitor', 'inductor'. This controls the initial condition group box design

Index	Description
3	Initial condition enabled for operating point check box. ('true' or 'false')
4	Initial condition enabled fro transient check box. ('true' or 'false')
5	Initial condition enabled for AC check box. ('true' or 'false')
6	Initial condition enabled for DC check box. ('true' or 'false')

Argument 3

String array defining list of parameter names. See argument 4.

Argument 4

String array defining list of parameter values. If arguments 3 and 4 are supplied the Parameters... button will be visible. This button opens another dialog box that provides the facility to edit these parameters' values.

Return value

The function returns a string array in the following form

Index	Description
0	Value in Result box
1	Value in Initial Voltage or Initial Current box. Empty if Open circuit or Short circuit is selected
2	Number of parameter values.
3 onwards	The values of the parameters in the order they were passed.
number of parameters +3	Initial condition enabled for operating point check box. ('true' or 'false')
number of parameters +4	Initial condition enabled fro transient check box. ('true' or 'false')
number of parameters +5	Initial condition enabled for AC check box. ('true' or 'false')
number of parameters +6	Initial condition enabled for DC check box. ('true' or 'false')

EditSelect

Type	string	string	string
Description	initial edit control entries	labels	dialog box caption
Compulsory	No	No	No
Default	<<empty>>	<<empty>>	<<empty>>

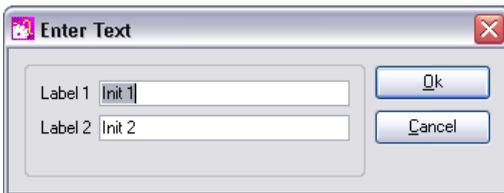
Return type: string array

Opens a dialog box containing any number of *edit controls* allowing the user to enter text values. The number of edit controls is the smaller of the lengths of arguments 1 and 2. If no arguments are given, 6 controls will be displayed with blank labels. Function returns string vectors containing user entries for each control. If cancel is selected, a single empty string is returned.

Example

The following dialog box will be displayed on a call to

```
EditSelect(['Init 1','Init 2'],['Label 1','Label 2'],'Enter Text')
```



See Also

BoolSelect [page 79](#)
 RadioSelect [page 265](#)
 ValueDialog [page 316](#)

EditTimer

Type	real	string	real
Description	Timer ID	action	Value
Compulsory	Yes	Yes	No
Default			

Return type: real

Edit a timer. The function can stop a timer or change its interval. To delete a timer, use the [DeleteTimer \(page 110\)](#) function.

Argument 1

Timer ID as returned by the [CreateTimer \(page 93\)](#) function

Argument 2

Action. This can be either:

1. 'interval' in which case this function will change the interval of the timer identified in argument 1 to the value specified in argument 3
2. 'kill' in which case the timer will be stopped. The timer will not be deleted and can be restarted by calling this function with the 'interval' action

Argument 3

Required if 'interval' is specified in argument 2

Return Value

Returns 1.0 if the function is successful. Otherwise returns 0.0. The function will fail if the specified timer does not exist, if the action is not recognised or if the action is 'interval' and argument 3 is not specified.

EditWaveformDialog

Type	real	real
Description	Time/frequency initial values	Vertical initial values
Compulsory	Yes	No
Default		

Return type: String array

Opens the dialog box shown below designed for editing a time domain waveform. To see an example of this dialog box, place a Waveform Generator on a schematic, select it then press F7.

Argument 1

Initial values for the controls in the Time/Frequency group box. Up to 10 elements defined as follows:

Index	Description
0	Integer from 0 to 8, specifies wave shape as follows: 0 Square 1 Triangle 2 Sawtooth 3 Sine 4 Cosine 5 Pulse 6 One pulse 7 One pulse (exp) 8 Step
1	Delay
2	Rise time
3	Fall time
4	Width
5	Period
6	Damping
7	0: Use Delay, 1: Use Phase
8	Frequency
9	Duty cycle

Argument 2

Initial values for the controls in the Vertical group box. Up to 5 elements defined as follows:

Index	Description
0	Initial
1	Pulse
2	Off until delay
3	Offset
4	Amplitude

Return value

String array with 15 elements. Elements 0 - 9 as for argument 1, elements 10-14 as for argument 2

EnterTextDialog

Type	string array
Description	initial text and box caption
Compulsory	Yes
Default	

Return type: string

Opens a dialog box allowing the user to enter lines of text. The argument specifies the initial text and the dialog box's caption as follows:

0	Initial text
1	Dialog box caption

The function returns the text entered by the user

EscapeString

Type	string
Description	string
Compulsory	Yes
Default	

Return type: string

Process string to replace escaped characters with literals.

Argument 1

Input string

Return value

Returns the input string but with the following character sequences substituted with their literal values as follows:

\t	Replaced with a tab character. (ASCII code 9)
\n	Replaced with a new line character. (ASCII code 10)
\r	Replaced with a carriage return character. (ASCII code 13)
\f	Replaced with a form feed character. (ASCII code 12)

\\ Replaced by a single '\'
 '\ ' followed by any other character
 Replaced by the character following the '\ '. The '\ ' itself is omitted.

ev

Type	any type	any type	... Up to 8 arguments in total
Description	vector	vector	
Compulsory	Yes	No	
Default			

Return type: real/complex array

Special function used to evaluate a sequence of expressions without requiring multiple Let statements. Useful for schematic TEMPLATES and similar.

Arguments

This function may be supplied with up to 8 arguments. All arguments except the last is ignored by the function.

Return value

The function returns the value of the last argument supplied

Notes

The purpose of this function is to allow the evaluation of intermediate variables withing a single expression. This is useful when the expression is in a schematic or graph template, for example, where there is only the facility available to enter a single expression.

For example:

```
ev (x=3, x*x)
```

returns 9. The first argument is evaluated and assigns 3 to x. The second argument is then evaluated using the value of x assigned in argument 1. In a script, it would be more conventional to use the 'Let' command to assign x. But if the expression was used in a template property, there is no facility to execute commands, so this would not be possible.

Execute

Type	string	any	May have up to 8 args in total
Description	Script name	Script argument 1	Script args 2 - 7
Compulsory	Yes	No	No
Default			

Return type: Depends on called script

Function calls the script defined in arg 1 and passes it the arguments supplied in arg 2-8. The function's returned value is the script's first argument passed by reference. The Execute function is used internally to implement user functions that are registered with the RegisterUserFunction command. See [“User Defined Script Based Functions” on page 461](#).

ExistDir

Type	string
Description	directory name
Compulsory	Yes
Default	

Return type: real

Function returns a real scalar with one of three values:

0	Directory does not exist
1	Directory exists with write privilege
2	Directory exists but with no write privilege

ExistFunction

Type	string	string
Description	Function name	Function type
Compulsory	Yes	No
Default		'global'

Return type: real

Returns TRUE or FALSE depending on whether specified function exists.

Argument 1

Function name.

Argument 2

Either 'global' or 'script'. If 'global', arg 1 is assumed to be the name of a built in function. If 'script' arg 1 is assumed to be a function defined as a script and installed using the [RegisterUserFunction](#) (page 408).

User defined compiled functions linked in as a DLL/shared library are treated as 'global'.

ExistSymbol

Type	string	string
Description	symbol name	scope
Compulsory	Yes	No
Default		'global'

Return type: real

Returns TRUE if symbol name given in argument 1 exists. Argument 2 specifies the scope of the search. If set to 'global', only the global library will be searched, if set to 'local', only the current schematic's local symbols will be searched. If set to 'all', both will be searched.

ExistVec

Type	string	string
Description	vector name	options
Compulsory	Yes	No
Default		

Return type: real

Returns TRUE (1) if the specified vector exists otherwise returns FALSE (0). If the second argument is 'GlobalLocal', only the global and local groups are searched for the vector otherwise the current group is also searched. See [“Groups” on page 37](#) for further details.

exp

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Returns e raised to the power of argument. If the argument is greater than 709.016, an overflow error occurs.

fft

Type	real array	string
Description	vector	window function
Compulsory	Yes	No
Default		'Hanning'

Return type: complex array

Performs a Fast Fourier Transform on supplied vector. The number of points used is the next binary power higher than the length of argument1. Excess points are zero-filled. Window used may be 'Hanning' (default) or 'None'.

User's should note that using this function applied to raw transient analysis data will not produce meaningful results as the values are unevenly spaced. If you apply this function to simulation data, you must either specify that the simulator outputs at fixed intervals (select the Output at .PRINT step option in the Simulator|Choose Analysis... dialog box) or you must interpolate the results using the Interp function - see [page 223](#). The FFT plotting menu items run a script which interpolate the data if it detects that the results are unevenly spaced. Use of these menus does not require special consideration by the user.

Field

Type	real	real	real
Description	Value	first bit	second bit
Compulsory	Yes	Yes	Yes
Default			

Return type: real

Function provides bit access to integers. Returns the decimal value of a binary number composed from the binary representation of argument 1 between the bit numbers defined in arguments 2 and 3. E.g.:

```
Field(100, 1, 3) = 2  
  
100 (decimal) = 1100100 (binary)  
bits 1 to 3 (from right i.e. least significant) = 010 (binary)  
= 2
```

Field is useful for cracking the individual bits used for symbol attribute flags. See [“Attribute flags” on page 404](#).

FilterFile

***** UNSUPPORTED FUNCTION *** see page 72**

Type	string	string array	string
Description	file name	Keywords	Option
Compulsory	Yes	Yes	No
Default			

Return type: string array

Processes a file specified by arg 1 and returns a string array containing any lines in the file that start with any of the keywords specified by arg 2. If arg 3 = 'strip', the lines will be returned with the keyword removed.

If arg3='spice', the input file will be filtered to remove inline comments and join lines connected using the '+' continuation character. Note that with arg3='spice' normal '*' comments pass through unmodified as long as they are not embedded between '+' continuation lines.

This function was developed for internal testing and was used to extract control lines from netlists. It may have other uses.

FindModel

Type	string	string	string
Description	model name	model type (e.g. 'Q' for BJT, 'X' for subcircuit)	Simulator type
Compulsory	Yes	Yes	No
Default			'SIMetrix'

Return type: string array

Returns string array of length 2 holding the file name and line number of the definition of the specified model. Set argument 3 to 'SIMPLIS' to search for a SIMPLIS model.

FIR

Type	real array	real array	real array
Description	vector to be filtered	filter coefficients	initial conditions
Compulsory	Yes	Yes	No
Default			All zero

Return type: real array

Performs “Finite Impulse Response” digital filtering on supplied vector. This function performs the operation:

$$y_n = x_n \cdot c_0 + x_{n-1} \cdot c_1 + x_{n-2} \cdot c_2 + \dots$$

Where:

x is the input vector (argument 1)

c is the coefficient vector (argument 2)

y is the result (returned value)

The third argument provide the ‘history’ of x i.e. x_{-1} , x_{-2} etc. as required.

The operation of this function (and also the function “**IIR**” on page 215) is simple but its application can be the subject of several volumes! Below is the simple case of a four sample rolling average. In principle an almost unlimited range of FIR filtering operations may be performed using this function. Any text on Digital Signal Processing will provide further details.

User's should note that using this function applied to raw transient analysis data will not produce meaningful results as the values are unevenly spaced. If you apply this function to simulation data, you must either specify that the simulator outputs at fixed intervals (select the Output at .PRINT step option in the Simulator|Choose Analysis... dialog box) or you must interpolate the results using the Interp function - see [page 223](#)

Example

Suppose a vector VOUT exist in the current group (simulation results). The following will plot VOUT with a 4 sample rolling average applied

```
Plot FIR(vout, [0.25, 0.25, 0.25, 0.25])
```

Alternatively, the following does the same

```
Plot FIR(vout, 0.25*unitvec(4))
```

Floor

Type	real
Description	scalar
Compulsory	Yes
Default	

Return type: real

Returns the argument truncated to the next lowest integer. Examples

```
Floor(3.45) = 3  
Floor(7.89) = 7  
Floor(-3.45) = -4
```

This function accepts only scalar input values. See Floorv below for a version that accepts vector input.

Floorv

Type	real
Description	vector
Compulsory	Yes
Default	

Return type: real

Returns the argument truncated to the next lowest integer. Same as Floor above but also accepts vector inputs. E.g.

```
Floorv([3.45, 7.89, -3.45]) = [3, 7, -4]
```

FormatNumber

Type	real	real	string
Description	number	significant digits	format
Compulsory	Yes	Yes	No
Default			'eng'

Return type: real

Formats a real value and returns a string representation of it. Argument 2 is the number of significant digits and argument 3 specify what format to use. The options are:

'eng'	(default if omitted). Formats the number using engineering units
'noeng'	Normal format. Will use 'E' if necessary
'%'	Formats as a percentage

Fourier

Type	real array	real	real	real array
Description	data	Fundamental frequency	Number of frequency terms	options
Compulsory	Yes	Yes	Yes	No
Default				

Return type: complex array

Calculates the fourier spectrum of the data in argument 1. The function uses the 'Continuous Fourier' technique which numerically integrates the Fourier integral. Because this technique does not require the input data to be sampled at evenly spaced points, it doesn't suffer from frequency aliasing. This is the main drawback of the more commonly used FFT (Fast Fourier Transform) algorithm. However, the Continuous Fourier algorithm is much slower than the FFT, sometimes dramatically so.

Argument 1

The input data. This is expected to possess a reference i.e. x-values

Argument 2

Specifies the fundamental frequency. All terms calculated will be an integral multiple of this.

Argument 3

Specifies the number of frequency terms to be calculated.

Argument 4

This is optional and can be a 1 or 2 element array. The first element is the first frequency to be calculated expressed as a multiple of the fundamental. The default value is 0 i.e. the DC term is calculated first. The second element is the integration order used and may be 1 or 2.

Return Value

The result of the calculation and will be a complex array with length equal to argument 3.

FourierOptionsDialog

Type	string array	real array
Description	initial values	sample vector
Compulsory	Yes	No
Default		

Return type: string array

Same as DefineFourierDialog except that only the Fourier sheet is displayed. The remaining tabbed sheets are hidden.

FourierWindow

Type	real	string
Description	Input vector	window type
Compulsory	Yes	No
Default		'hanning'

Return type: real array

Returns the input vector multiplied by one of a selection of 4 window functions. This is intended to be used with a Fourier transform algorithm.

Argument 1

Input vector

Argument 2

Window type. One of:

'hanning'
'hamming'
'blackman'
'rectangular'

FullPath

Type	string	string
Description	relative path name	reference directory
Compulsory	Yes	No
Default		Current working directory

Return type: real

Returns the full path name of the specified relative path and reference directory.

Examples

```
FullPath('amplifier.sch', 'c:\simulation\circuits') =  
c:\simulation\circuits\amplifier.sch
```

```
FullPath('../amplifier.sch', 'c:\simulation\circuits') =  
c:\simulation\amplifier.sch
```

See also

RelativePath [page 272](#)

SplitPath [page 297](#)

GenPrintDialog

Type	string array	string
Description	initial settings	Enabled modes
Compulsory	Yes	No
Default		

Return type: string array

Opens a dialog box used to define print settings

Argument 1

The argument is a string array of length 13 and defines the initial settings of the dialog box as follows:

Index	Description
0	'area' "Fit Area" 'grid' "Fixed Grid"
1	Schematic scale (entered as a string)
2	Schematic caption
3	Graph magnification (entered as a string)
4	Graph caption
5	Orientation 'landscape' or 'portrait'
6	Layout: '0' Schematic only '1' Graph only '2' Schematic/Graph '3' Graph/Schematic
7	Left margin. The value is entered and returned in units of 0.1mm but will be displayed according to system regional settings. Must be entered as a string
8	Top margin. Comments as for left margin.
9	Right margin. Comments as for left margin.
10	Bottom margin. Comments as for left margin.
11	Major grid checked: 'on' Checked 'off' Not checked
12	Minor grid checked: 'on' Checked 'off' Not checked

Argument 2

Specifies whether schematic mode, graph mode or both are enabled. If omitted the mode is determined by the schematic and graph windows that are open.

To enable schematic mode only, set this argument to 'Schem', to set to graph mode set to 'Graph' and to set to both, set to 'Schem|Graph'.

Return value

The function returns a string array with the same format as argument 1 and assigned with the user's settings. If the user selects Cancel the function returns an empty vector.

GetActiveWindow

No arguments

Return type: string array

Return details about currently active window

Return value

Returns string array of length 2 providing the following details about the currently active window:

Index	Description
0	Type of window. Maybe one of: CommandShell Schematic Graph Symbol
1	Title of window as displayed in the title bar

GetAllCurves

No arguments

Return type: string array

Returns an array listing id's for all curves on currently selected graph. All curves are referred to by a unique value that is the 'id'. Some functions and command require a curve id as an argument.

GetAllSymbolPropertyNames

Type	string
Description	Options
Compulsory	No
Default	'none'

Return type: string array

Returns a string array containing the names of all the properties on the symbol currently open in the symbol editor.

Argument 1

Options. Currently, there is only one which is 'nopins'. If *not* present, the function will return all properties including the internally generated properties used to display pin names. These are of the form \$Pin\$pinname. If 'nopins' is specified, these properties will not be returned by the function.

GetAllYAxes

No arguments

Return type: string array

Returns an array listing all y axis id's for currently selected graph. All graph axes have a unique 'id' which may be used with some other commands and functions.

GetAnalysisInfo

Type	string
Description	Options
Compulsory	No
Default	

Return type: string array

Returns the parameters of the most recent analysis performed by the simulator. The parameters are returned in the form of a string array. If argument 1 is set to 'name' the function will return the names of each parameter.

The following sample shows how to obtain a the stop time of a transient analysis:

```
let stopIdx = Search(GetAnalysisInfo('name'), 'tstop')
Let stopTime = Val(info[stopIdx])
```

The following table shows the parameter names currently available for each analysis type:

Analysis Type	Parameter Names
Transient	ANALYSISNAME, GROUPNAME, TSTART, TSTOP, TSTEP, TMAX, UIC, DELTA, RTNSTART, RTNSTOP, RTNSTEP, RTNENABLED, FAST
AC	ANALYSISNAME, GROUPNAME, PARAM, MODEL, TEMP, FREQ, MONTE, REPEAT, DEVICE, MODE, START, STOP, STEP, NUMSTEPS, GRAD, SINGLE, F
DC	ANALYSISNAME, GROUPNAME, PARAM, MODEL, TEMP, FREQ, MONTE, REPEAT, DEVICE, MODE, START, STOP, STEP, NUMSTEPS, GRAD, SINGLE
Noise	ANALYSISNAME, GROUPNAME, PARAM, MODEL, TEMP, FREQ, MONTE, REPEAT, DEVICE, MODE, START, STOP, STEP, NUMSTEPS, GRAD, SINGLE, V, VN, I, INSRC, PTSPERSUM, F
Transfer Function	ANALYSISNAME, GROUPNAME, PARAM, MODEL, TEMP, FREQ, MONTE, REPEAT, DEVICE, MODE, START, STOP, STEP, NUMSTEPS, GRAD, SINGLE, V, VN, I, INSRC, F, IMODE
Sensitivity	ANALYSISNAME, GROUPNAME, POSNAME, NEGNAME, I, GRAD, START, STOP, NUMSTEPS
Pole-zero	ANALYSISNAME, GROUPNAME, NODEINAME, NODEGNAME, NODEJNAME, NODEKNAME, VOLCUR, POLZER
Operating point	ANALYSISNAME, GROUPNAME

GetAnalysisLines

No arguments

Return type: string array

Returns the analysis lines used in the most recent simulation analysis. The analysis lines are the lines in the netlist that specify an analysis such as `.tran`, `.ac` etc. The function will return an empty vector if no simulation has been run or if the latest run has been reset or was aborted.

GetAxisCurves

Type	string
Description	Y axis id
Compulsory	Yes
Default	

Return type: string array

Returns an array listing all curve id's for specified y-axis. All curves are referred to by a unique value that is the 'id'. Some functions and command require a curve id as an argument.

GetAxisLimits

Type	string
Description	Axis id
Compulsory	Yes
Default	

Return type: real array

Returns array of length 3 providing limits info for specified axis as follows:

0	Minimum limit
1	Maximum limit
2	Axis scale type - 0 = linear, 1 = logarithmic
3	Fixed or auto. 0 = fixed, 1 = auto

GetAxisType

Type	string
Description	Axis id
Compulsory	Yes
Default	

Return type: string

Returns string specifying type of axis. Possible values are:

'X'	X-axis
'Digital'	A Digital Y-axis. (Created with "Curve /dig" or menu Probe Voltage - Digital... .)
'Main'	Main Y-axis (axes at bottom of graph)
'Grid'	Grid Y-axis (axes stacked on top of main)
'NotExist'	Axis does not exist

GetAxisUnits

Type	string
Description	Axis id
Compulsory	Yes
Default	

Return type: string

Returns physical units of axis. See the function "[Units](#)" on page 313 for list of possible values.

GetChildModulePorts

Type	string	string	real
Description	Property name	Property value	Schematic handle
Compulsory	Yes	Yes	No
Default			-1 (use currently selected schematic)

Return type: string array

Finds information about module ports in the underlying schematic of a hierarchical block. This function was developed as part of the system to allow buses to pass through hierarchies as it can find whether the underlying module port for a hierarchical block is defined for bus connections.

Arguments 1 and 2

specify a property name and value that must uniquely define an instance. Usually arg 1 would be 'handle'. If arg 1 is an empty string, a single selected instance will be used.

Argument 3

Schematic handle as returned by the OpenSchematic function ([page 249](#)). This allows this function to be used with a schematic that is not open or not currently selected. If omitted or -1, the currently selected schematic will be used.

Return Value

String array of size = 2 X the number of module ports in the underlying schematic. Values arranged in pairs. The first in each pair in the name of the module port and the second value is the bus size. The latter will always be 1 for a non bus module port.

GetColours

No arguments

Return type: string array

Returns the names of all objects in the program whose colour may be edited. The function is usually used in conjunction the GetColourSpec function ([page 148](#)), the SelectColourDialog function ([page 281](#)) and the EditColour command ([page 377](#)).

GetColourSpec

Type	string
Description	Colour name
Compulsory	Yes
Default	

Return type: string

Returns the current colour specification for the object whose name is passed to argument 1. Valid object names can be obtained from the GetColours function ([page 148](#)). The return value may be used to initialise the SelectColourDialog ([page 281](#)) which allows the user to define a new colour.

The return value represents the colour of the object as a single integer which can be decoded into its RGB components. However, this value should not be used directly as its format may change in future versions of the product

If the object name passed is not recognised the function will return the representation for the colour black.

GetConfigLoc

No arguments

Return type: string array

Returns the location of the application's configuration settings. In versions prior to version 5, this would be in one of the following forms:

REG;*registry_root_pathname*

OR

PATH;*inifile_pathname*

If the first form is returned, the settings are stored in the registry the path being HKEY_CURRENT_USER*registry_root_pathname*

If the second form is returned the settings are stored in a file with full path equal to *inifile_pathname*.

From version 5, the registry is no longer used for storing settings, so only the second of the two forms will ever be returned.

The return value from GetConfigLoc can be used directly as the value of the /config_location switch at the simulator (SIM.EXE) command line. See the “Running the Simulator” chapter in the *Simulator Reference Manual* for more details.

GetConnectedPins

Type	real array	string	string
Description	schematic location	identifying property	pin number or pin name
Compulsory	Yes	No	No
Default		'ref'	'pinname'

Return type: string array

Function returns instance and pin name for all pins connected to net at specified point. Results are sorted according to the number of pins on owner instance.

Argument 1

Specifies a point on the schematic that identifies a net. This could be returned by the WirePoints function ([page 319](#)) for example.

Argument 2

Property whose value will be used to identify instance in returned values.

Argument 3

Specify whether pins to be identified by their name or number. If set to 'pinnumber', the number will be used otherwise the name will be used.

Return value

An array of strings of length equal to 2 times the number of pins on the net. The even indexes hold the property value identifying the instance and the odd indexes hold either the pin's name or number according to the value of argument 3.

Note that this function does not return pins on implicit connections. An implicit connection is one that is made by virtue of having the same netname as defined by a terminal symbol or similar but has no physical connection using wires.

Example

The following sequence will display the output of this function for a single selected wire on the schematic:

```
** Get selected wires  
Let wires = SelectedWires()  
  
** Get locations for first wire in selected list  
Let points = WirePoints(wires[0])  
  
** Show connected pins  
Show GetConnectedPins([point[0], points[1]])
```

GetConvergenceInfo

No arguments

Return type: string array

Returns a string array providing convergence information about the most recent run. Each element of the array is a list of values separated by semi-colons. The output may be pasted into a spreadsheet program that has been set up to interpret a semicolon as a column separator. The first element of the array lists the names for each column and therefore provides a heading. The following headings are currently in use:

type	Node or Device
name	Name of node or device that failed to converge
count	Number of times node/device failed to converge during run
time (first step)	Time of most recent occurrence of a 'first step' failure.
required tol	Required tolerance for most recent 'first step' failure
actual tol	Tolerance actually achieved for most recent 'first step' failure
absolute val	Absolute value for most recent 'first step' failure
time (cut back step)	Time of most recent occurrence of a 'cut back step' failure.
required tol	Required tolerance for most recent 'cut back step' failure

actual tol	Tolerance actually achieved for most recent 'cut back step' failure
absolute val	Absolute value for most recent 'cut back step' failure
final?	Node or device failed on the final step that caused the simulation to abort
top analysis	Main analysis mode (Tran, DC etc.)
current analysis	Current analysis. Either the same as 'top analysis' or Op
op mode	Method being used for operating point. (PTA, JI2, GMIN or SOURCE)

A *first step* failure is a failure that occurred at the first attempt at a time step after a previously successful step. If a time point fails, the time step is cut back and further iterations are made. Failures on steps that have been cut back are referred to in the above table as *cut back steps*. Quite often the nodes or devices that fail on a *cut back step* are quite different from the nodes or devices that fail on a *first step*. The root cause of a convergence failure will usually be at the nodes or devices that fail on a *first step*.

It is quite difficult to interpret the information provided by this function. The 'where' script performs a simple analysis and sometimes displays the nodes or devices most likely to be the cause.

GetCurDir

No arguments

Return type: string

Returns current working directory.

GetCurrentGraph

No arguments

Return type: string

Returns id of the currently selected graph. Returns '-1' if no graphs are open. The id can be used in a number of functions that return information about graphs or graph objects generally.

GetCursorCurve

No arguments

Return type: string

Returns a string array of length 3 providing information on the curve attached to the measurement cursor. Returns an empty vector if cursors not enabled.

Index	Description
0	Curve id
1	Source group name. This is the group that was current when the curve was created.
2	Division index if curve is grouped. (E.g. for Monte Carlo)

GetCurveAxis

Type	string
Description	curve id
Compulsory	Yes
Default	

Return type: string

Returns the id of the y-axis to which the specified curve is attached.

GetCurveName

Type	string
Description	curve id
Compulsory	Yes
Default	

Return type: string

Returns name of specified curve

GetCurves

No arguments

Return type: string array

Returns an array of curve names (as displayed on the graph legend) for the current graph.

GetCurveVector

Type	real	real	string
Description	curve id	Division index	Obsolete - no longer used
Compulsory	Yes	No	No
Default		Return all divisions	

Return type: real array

Returns the data for a curve.

For a single curve (i.e. not a group of curves as created from a Monte Carlo plot) only the first argument is required and this specifies the curve's id.

If the curve id refers to a group of curves created by a multi-step run, then the second argument may be used to identify a single curve within the group. The data for the complete curve set is arranged as a *Multi Division Vector* (see [page 38](#)). The second argument specifies the division index. If absent the entire vector is returned

Note that the arguments to this function for version 4 and later have changed from earlier versions.

The function `cv` is identical to `GetCurveVector` and is convenient in situations where a short expression is desirable.

GetDatumCurve

No arguments

Return type: string array

Returns a string array of length 3 providing information on the curve attached to the reference cursor.

Index	Description
0	Curve id
1	Source group name. This is the group that was current when the curve was created.
2	Division index if curve is grouped. (E.g. for Monte Carlo)

GetDeviceDefinition

Type	string	string	string	string
Description	Device name	Device type	Simulator type	Options
Compulsory	Yes	Yes	No	No
Default			'SIMetrix'	

Return type: string array

Searches for the specified device model in the global library and returns the text of the model definition. If the device is defined using a .MODEL control, the result will have a single element containing the whole definition. If the device is defined using a subcircuit then the result will be a string array with a single element for each line in the subcircuit definition.

Argument 1

The model/subcircuit name. E.g. 'Q2N2222' or 'TL072'

Argument 2

The type of the device. This may be either the device letter e.g. 'Q' for a BJT, or the model type name e.g. 'npn'. A list of device letters is given in the *Simulator Reference manual* in the “Running the Simulator” chapter.

If the device is a subcircuit, use the letter 'X'.

Argument 3

This must be either 'SIMetrix' or 'SIMPLIS'. If set to SIMPLIS, only subcircuits declared for use with SIMPLIS will be returned. This is done using the .SIMULATOR control in the library file. Note that only SIMPLIS subcircuits are supported. Currently SIMPLIS devices defined using .MODEL are not supported by the SIMetrix model library manager.

Argument 4

Options. Currently there is only one: set this argument to 'header' to instruct the function to output preceding comment text. If this is set, up to 20 comment lines (starting with ‘*’) before the start of the model will also be output.

GetDeviceInfo

Type	string	string
Description	Model name	Options
Compulsory	Yes	No
Default		none

Return type: string array

Returns information about the specified simulator device.

Argument 1

Internal device name as returned by the GetModelType or GetInternalDeviceName function. This is not the same as the type name used in the .MODEL control but a name that is used internally by the simulator. For example, the internal device name for a LEVEL 1 MOSFET is 'MOS1'.

Optionally the device letter may be specified if `arg2 = 'letter'`. However, the function will not return such precise information if this option is used. For example, the LEVEL value will not be known and so -1 will be returned. Also the minimum and maximum number of terminals will reflect all devices that use that device letter and not just one specific device. E.g. the 'BJT' device defines the standard SPICE Gummel-Poon transistor which can have 3 or 4 terminals. But the 'q' letter can also specify VBIC_Thermal devices which can have 5 terminals.

Argument 2

Options, currently only one. If this is set to 'letter', a single letter should be specified for argument 1. This is the device letter as used in the netlist, e.g. 'Q' for a BJT, 'R' for a resistor. See notes above concerning specifying using the device letter.

Return Value

Result is a 6 element array. Each element is defined as follows:

Index	Description
0	Model type name for negative polarity device. E.g. 'npn', 'nmos' etc.
1	Model type name for positive polarity device E.g. 'pnp', 'pmos' etc. Empty if device has only a single polarity
2	Device letter. E.g. 'Q' for a BJT

Index	Description
3	Maximum number of terminals.
4	Minimum number of terminals. This is usually the same as the maximum number of terminals, except for BJTs whose substrate terminal is optional.
5	Value required for LEVEL parameter. 0 means that this is the default device when no LEVEL parameter is specified. -1 will be returned if the 'letter' option is specified.
6	Semi-colon delimited list of valid .MODEL control model name values. E.g. 'nnp', 'pnp' and 'lpnp' are returned for the 'BJT' device.

GetDeviceParameterNames

Type	string	real	string array
Description	device type	level	Options
Compulsory	Yes	No	No
Default		-1	

Return type: string array

Returns string array containing all device parameter names for the specified simulator model type.

Argument 1

Device type specified using its *SPICE letter* e.g. 'Q' for a BJT, 'M' for a MOSFET etc.

Argument 2

Model level if relevant. If omitted or set to -1, the default level for that type of device will be used.

Argument 3

String array of length up to 2. May contain one or both of 'useInternalName' and 'readback'. If 'useInternalName', then argument 1 must specify the device's internal name. This is returned by GetInternalDeviceName ([page 171](#)). Argument 2 is ignored in this case.

If 'readback' is specified, the function returns names of 'read back' parameters. Read back parameters aren't writeable but return information about a device's operating characteristics. For example, most MOS devices have 'vdsat' read back parameter that returns the saturation voltage. This function only returns the names of read back parameters. To find their values, use GetInstanceParamValues ([page 169](#)).

Return value

String array of length determined by the number of parameters the device has. Each element contains the name of a single parameter. To find the values for the parameters use `GetInstanceParamValues` ([page 169](#)).

Example

The following:

```
Show GetDeviceParameterNames('M')
```

returns:

```
0   'L'
1   'W'
2   'M'
3   'AD'
4   'AS'
5   'PD'
6   'PS'
7   'NRD'
8   'NRS'
9   'IC-VDS'
10  'IC-VGS'
11  'IC-VBS'
12  'TEMP'
```

GetDotParamNames

No arguments

Return type: string array

Returns names of variables defined using `.PARAM` in the most recent simulation run.

Return Value

String array with names of variables. If no simulation has been run, an empty result will be returned. Note that real values in the front end's global group are passed to the simulator and entered as `.PARAM` values. So this function will always return those values. In addition the values 'PLANCK', 'BOLTZ' and 'ECHARGE' are always defined.

GetDotParamValue

Type	string
Description	Variable name
Compulsory	Yes
Default	

Return type: real

Returns the value of a variable defined using .PARAM in the most recent simulation run

Return Value

Real value of variable. If variable does not exist or if no simulation has been run, an empty result will be returned.

GetDriveType

Type	string
Description	path
Compulsory	Yes
Default	

Return type: string

Determines the type of drive or file system of the specified path. Returns one of the following values:

Return value	Description
'local'	Drive or file system present on the local machine
'remote'	Network drive or file system
'cdrom'	CD Rom or DVD drive
'other'	Other file system or drive. (A RAM drive on Windows will return this type, as will /proc on Linux)
'notexist'	The path doesn't exist or media not present.
'unknown'	Drive type or file system could not be determined

Notes

This function is not reliable on Linux systems. It works by matching the file system type to known file systems but if the file system is not recognised, it will return 'other'.

GetEmbeddedFileName

Type	string
Description	file path
Compulsory	Yes
Default	

Return type: string

Returns the actual file name used for an embedded file specified using ‘.FILE’ and ‘.ENDF’.

Argument 1

Name of embedded file. That is the name used after .FILE

Return value

‘.FILE’ and ‘.ENDF’ allow file to be embedded in netlist and this is implemented by writing the contents to a real file. This function returns the full path name of the real file

Notes

This function can be used to access an embedded file in a script called using the .POST_PROCESS statement. This is useful, for example, to embed data in a netlist to be accessed in the .POST_PROCESS script.

This function may also be called after a simulation has been run to access data contained in any .FILE/.ENDF block.

GetEnvVar

Type	string
Description	system environment variable name
Compulsory	Yes
Default	

Return type: string

Returns the value of a system environment variable.

GetEthernetAddresses

No arguments

Return type: string array

Returns information about the installed Ethernet adapters.

Return Value

Returns a string array providing information about the Ethernet adapters installed in the system. Depending on the operating system, this will either be a simple list of Ethernet addresses or a list of semi-colon delimited strings providing the Ethernet address followed by a description of the adapter.

GetF11Lines

Type	string	real
Description	Options	Schematic handle
Compulsory	No	No
Default		-1 (I.e. use selected schematic)

Return type: string array

Returns the contents of the schematic's text window also known as the F11 window. Each element of the returned array contains a single line of the F11 text.

Argument 1

If set to 'spice' the lines will be filtered to remove inline comments and join lines connected using the '+' continuation character. Note that with arg1='spice' normal '*' comments pass through unmodified as long as they are not embedded between '+' continuation lines. Also, leading spaces will also be stripped in this mode.

Argument 2

Schematic handle as returned by [OpenSchematic \(page 249\)](#). This makes it possible to apply this function to any schematic and not just the one that is currently displayed. See "[OpenSchematic](#)" on [page 249](#) for more details.

GetFile

Type	string	real
Description	File specification	0: file must exist, 1: need not exist
Compulsory	Yes	No
Default		0

Return type: string

Opens the Open File dialog box. Return value is full pathname of file selected by user. If user cancels operation, function returns an empty string. Argument to function supplies description of files and default extension. These two items are separated by '\'. E.g. `getfile('Schematic Files\sch')` .

This function has now been superseded by the functions `GetSimetrixFile` ([page 189](#)) and `GetUserFile` ([page 205](#)) which are more flexible.

GetFileCd

This function is now obsolete. Use the functions `GetSimetrixFile` ([page 189](#)) or `GetUserFile` ([page 205](#)) instead.

GetFileExtensions

Type	string
Description	file type
Compulsory	Yes
Default	

Return type: string array

Returns a string array containing all valid extensions (without prefixed '!') for the given file type. The extension returned in the first element is the default. File extensions can

be changed in the general options dialog box (File|Options|General...) and are stored in a number of option variables. These are listed in the following table .

Argument	Used for	Option name	Default
'Schematic'	Schematic files	SchematicExtension	sxsch, sch
'Data'	Data files	DataExtension	sxdat, dat
'Text'	Text files	TextExtension	txt, net, cir, mod, ldf, sxscr, lib, lb, cat
'Symbol'	Binary symbol files	SymbolExtension	lib
'LogicDef'	Logic definition files used with arbitrary logic block	LogicDefExtension	ldf
'Script'	Script files	ScriptExtension	sxscr, txt
'Model'	Model files	ModelExtension	lb, lib, mod, cir
'Catalog'	Catalog files	CatalogExtension	cat
'Graph'	Graph binary files	GraphExtension	sxgph

GetFileInfo

Type	string
Description	file path
Compulsory	Yes
Default	

Return type: string array

Returns information about a specified file.

Argument 1

File path.

Return Value

Returns an array of length 5 containing the following information

Element Index	Description
0	Drive type, one of: 'local' 'cdrom' 'remote' 'other' 'notexist' 'unknown' This is not reliable on Linux systems. See notes for function GetDriveType (page 158)
1	File size in bytes
2	Full path name
3	Last modified time. Value is number of seconds elapsed since January 1, 1970
4	'True' if path is a directory, otherwise 'false'

GetFileSave

This function is now obsolete. Use [GetSimetrixFile \(page 189\)](#) or [GetUserFile \(page 205\)](#) instead.

GetFonts

No arguments

Return type: string array

Returns the names of all objects in the program whose font may be edited. The function is usually used in conjunction the function [GetFontSpec \(page 163\)](#), the function [SelectFontDialog \(page 286\)](#) and the command [EditFont \(page 378\)](#).

GetFontSpec

Type	string
Description	Object name
Compulsory	Yes
Default	

Return type: string

Returns the current font specification for the object whose name is passed to argument 1. Valid object names can be obtained from the GetFonts function ([page 163](#)). The return value may be used to initialise the SelectFontDialog ([page 286](#)) which allows the user to define a new font.

The return value represents the font of the object as a string consisting of a number of values separated by semi-colons. The values define the font in terms of its type face, size, style and other characteristics. However, these values should not be used directly as the format of the string may change in future versions of the product. The return value should be used only as an argument to functions or commands that accept a font definition. E.g. The SelectFontDialog ([page 286](#)) function and EditFont command ([page 378](#))

If the object name passed is not recognised the function will return the definition for the default font.

GetFreeDiskSpace

Type	string
Description	Directory
Compulsory	Yes
Default	

Return type: real

Returns free space on disk volume holding specified file or directory

Argument 1

A file or directory that resides on the disk volume whose free space is required. On windows this may be simply the drive letter followed by a colon. E.g. 'C:'

Return Value

Free space available in bytes

GetGraphObjects

Type	string	string
Description	Object type name	Graph id
Compulsory	No	No
Default		

Return type: string array

See [page 448](#) for information on graph objects

Returns a list of IDs for the graph objects defined by the optional arguments as follows:

If no arguments are specified, the IDs for all graph objects are returned.

If the first argument is specified, all objects of the defined type will be returned

If both arguments are specified, all objects of the defined type and located on the specified graph will be returned.

If the type name is invalid, or if the graph id specified in arg 2 is invalid or if there are no graphs open, the function will return an empty vector.

GetGraphObjPropNames

Type	string
Description	Graph object ID
Compulsory	Yes
Default	

Return type: string array

See [page 448](#) for information on “graph objects”

Returns the valid property names for the graph object defined by argument 1

GetGraphObjPropValue

Type	string	string
Description	Graph object ID	Property name
Compulsory	Yes	No
Default		Return all values

Return type: string array

See [page 448](#) for information on “graph objects”

Returns property values for the specified object. If argument 2 is present the value of one particular property will be returned. Otherwise the function will return an array containing all property values. The order of the values corresponds to the return value of [GetGraphObjPropNames](#) ([page 165](#)).

(Note the function `GetGraphObjPropValues` is the same but will only accept one argument)

GetGraphTabs

Type	real
Description	Graph window index
Compulsory	No
Default	Selected window

Return type:string array

Returns the graph IDs of all tabbed sheets in the specified graph window

Argument 1

Index of graph window. This is returned as the 'user index' field by the function [GetWindowNames \(page 208\)](#) with the 'full' option specified. If omitted, the function will operate on the currently selected graph window

Return Value

Returns an array of strings of length equal to the number of tabbed sheets in the selected graph window. Each element in the array is the ID of the graph object displayed in the tabbed sheet. The ID may be used in functions such as [GetGraphObjPropValue \(page 165\)](#) to obtain information about the graph including curves, axes, titles etc.

GetGraphTitle

No arguments

Return type:string

Returns title of currently selected graph.

GetGroupInfo

Type	string
Description	group name
Compulsory	Yes
Default	

Return type: string array

Returns information about a group.

Argument 1

Group name for which information is required. Enter “” to obtain information on the current group.

Return Value

String array of length 3 as described in the following table:

Index	Description
0	Source file. This is the path name for the file that contains the data for the group. If the groups data is stored in RAM, this element will hold an empty string
1	Group title. For groups created by a simulation (which is to say virtually all groups) this is obtained from the netlist title
2	Empty - reserved for future use

For more information on groups, see “Groups” on page 37

GetGroupStepParameter

Type	string
Description	Group name
Compulsory	No
Default	Current group

Return type: string

Returns the name of the ‘stepped parameter’ of a multi-step run. This value is stored within the group created for the simulation run’s output data. The stepped parameter is a label that identifies the parameter, device, model parameter or other quantity that is varied during a multi-step run.

GetGroupStepVals

Type	string
Description	Group name
Compulsory	No
Default	Current group

Return type: real array

Returns the 'stepped values' in a multi-step run. These values are stored within the group created for the simulation run's output data. The stepped values are the values assigned to the 'stepped parameter' (see [GetGroupStepParameter](#) function above) during a multi-step run.

GetInstanceBounds

Type	string	string	string
Description	Property name	Property value	Options
Compulsory	Yes	Yes	No
Default			none

Return type: real array

Returns the bounds occupied by a schematic instance identified by a property name and value.

Argument 1

Property name used to identify device. The 'Handle' property can be used to uniquely identify any schematic instance.

Argument 2

Value required for property identified in arg 1.

Argument 3

If set to 'body', the function will return the bounds of the graphics of the symbol only. This excludes the area occupied by any displayed properties. If this is omitted the bounding area returned will include all *visible* property text.

Return Value

The function returns a four element real array which defines the area occupied by the instance. The values are in "sheet units". There are 120 sheet units per visible grid square at X 1 magnification. The four elements of the array are in the order top, left, right, bottom. Values increase left to right and top to bottom.

GetInstanceParamValues

Type	string	string	string
Description	Instance name	Parameter name	Options
Compulsory	Yes	No	No
Default		Get all parameters	

Return type: string or string array

Returns simulation instance parameter values for the device specified. This function returns the values used in the most recent simulation. If simulation has been run, or it was aborted or reset (using Reset command), then this function will return an empty vector.

If argument 3 is set to 'readback', this function will return the values for readback parameters.

Argument 1

Instance name, e.g. Q23, R3 etc. This is the name used in the netlist stripped of its dollar prefix if applicable.

Argument 2

Name of parameter whose value is required. If this argument is missing or empty, then all parameters will be returned. The number and order of the parameters in this case will match the return value of parameter names from the function `GetDeviceParameterNames` ([page 156](#)).

Argument 3

If set to 'readback' and argument 2 is empty, this function will return the values of all read back values for the devices. 'read back' values are values calculated during a run and give useful information about a device's operating conditions. Note that the value returned will reflect the state of the device at the last simulation point. For example, if a transient run has just been performed, the values at the final time point will be given. If a small-signal analysis has been performed, the results will usually reflect the DC operating point conditions.

Return Value

If argument 2 is provided and valid, will return a single string expressing the value of the parameter. If arg 2 is missing or empty, a string array will be returned with all parameter values.

GetInstancePinLocs

Type	string	string	string	real
Description	property name	property value	options	Schematic handle
Compulsory	No	No	No	No
Default			'relative'	-1 meaning use selected schematic

Return type: real array

Return an array of pin locations for the symbol identified by arguments 1 and 2.

Argument 1, 2

Property name and value to identify instance. If these arguments are not supplied, the selected instance, if any, will be used instead. If there are no selected instances or no instances that match the arguments, the function will return an empty vector. If the arguments identify more than one instance, the function will return information for one of them but there are no rules to define which one.

Argument 3

If set to 'absolute', the values returned will be relative to a fixed origin on the schematic. Otherwise they will be relative to the origin of the instance. The origin of an instance can be determined using the InstPoints function ([page 220](#)).

Argument 4

Schematic handle as returned by the OpenSchematic function ([page 249](#)). This allows this function to be used with a schematic that is not open or not currently selected. If omitted or -1, the currently selected schematic will be used.

GetInstsAtPoint

Type	real array	string
Description	instance pin location	property name
Compulsory	Yes	Yes
Default		

Return type: real array

Functions finds the instances with pins at a specified point and returns a string array to identify them. The return value is a string array of length 2 X the number of pins at the

specified point. The first value in each pair is the value of the property identified in argument 2. The second value is the pin number (also referred to as the *netlist order*).

Argument 1 specifies the pin location and is the value returned from the `GetInstancePinLocs` ([page 170](#)) with the 'absolute' option specified.

GetInternalDeviceName

Type	string array
Description	Model details
Compulsory	Yes
Default	

Return type: string

Finds the simulator's internal device name for a model defined using its model type name and optionally, level and version.

The internal device name is a unique name used to define a primitive simulator device. For example, npn and pnp transistors have the internal device name of 'BJT'. Level 1 MOSFETs have the internal device name of 'MOS1' while nmos level 8 devices are called 'BSIM3'. Some functions - e.g. `GetDeviceInfo` ([page 155](#)) - require the internal device name as an argument.

Argument 1

1 - 3 element string array which describes device.

Index	Description
0	Model type name as used in the .MODEL control. E.g. 'nmos', 'npn' etc.
1	Optional. Value of LEVEL parameter. If omitted, default level is assumed
2	Optional. Value of VERSION parameter.

GetKeyDefs

No arguments

Return type: string array

Returns details of all key definitions. Note that only keys defined using `DefKey` are listed. Keys assigned as accelerators to menu definitions are not included.

Return Value

Returns an array of strings with each element in the array detailing a single key definition. Each definition is a semi-colon delimited string with three fields:

Field	Description
0	Name of key as entered in DefKey
1	Command executed by key press
2	Flag value. This is usually 4, but will be 5 for 'immediate' keys.

GetLastCommand

Type	string
Description	command group
Compulsory	Yes
Default	

Return type: string

Retrieve last command issued by a menu or toolbar with a specified command group definition. This is used for operations such as "repeat last place".

Argument 1

Name of a command group. These are arbitrary strings that may be supplied to a DefMenu or DefButton command using the /comgroup switch.

Return value

If a menu or button defined with a /comgroup specification is executed, the command executed is stored. This function retrieves the most recent with the specified comgroup value.

Notes

Menus and buttons used for placing components on a schematic are defined using the comgroup value 'place'. So GetLastCommand('place') always returns the command used for the most recent place operation.

GetLastError

No arguments

Return type:string

Returns a string with one of three values signifying the status of the most recent command executed. The three values are:

'OK'	Command executed without error
'Error'	One or more errors occurred in the most recent command
'Fatal'	The most recent command was not recognised or the evaluation of a braced substitution failed.

The command switches `/noerr` and `/quiet` (see “Command Line Switches” in the “Command [Shell](#)” Chapter of the *User's Manual*) can be used to effectively disable non-fatal errors. This function allows customised action in the event of an error occurring. For example, if a simulation fails to converge, the `run` command yields an error. This function can be used to take appropriate action in these circumstances.

When a fatal error occurs, the command will abort unconditionally and this function returns 'Fatal'.

GetLegendProperties

Type	string	string
Description	curve id	options
Compulsory	Yes	No
Default		'names'

Return type: string array

Returns either all legend property names or all legend property values for specified curves. Legend properties are the text associated with curve names in the graphs legend panel. The legend panel is the area between the graph and the toolbar where the curve legends are located.

If argument 2 = 'values' the function returns legend property values. Otherwise it returns legend property names.

GetLibraryModels

Type	string	string
Description	Library spec	options
Compulsory	Yes	No
Default		

Return type: string array

Returns a string array containing information about each model in the specified model library.

Argument 1

Library specification for installed library. This could be a single file or a folder containing a wildcard specification. All installed libraries are returned by [GetModelFiles \(page 177\)](#).

Argument 2

If set to 'usermodelonly' only models installed by the user will be returned.

Return value

String array with each element describing a single library model. Information is supplied as a semi-colon delimited string with the following fields:

Field Index	Description
0	Model name
1	File where model found. (Filename only, not full path)
2	Line number
3	SPICE letter. E.g. 'x' for subcircuits
4	Is alias: 'false' not an alias, 'true' is an alias
5	User install time. 0 if system installed. Time is number of seconds since January 1, 1970

GetLicenseInfo

No arguments

Return type: string array

Returns information about the current license.

Return value

String array as defined in the following table:

Index	Description
0	License type. One of 'Network', 'NamedUser', 'Nodelocked', 'Portable' or 'Unknown'
1	License serial number
2	Licensee
3	License location. Server name if network.
4	Additional information specific to license type. For portable licenses this is the type and serial number of the hardware key. (dongle)
5	Expiry date. Returns 'permanent' if non-expiring.
6	License version - this number is related to the maintenance expiry date.
7	Enabled features
8	Encryption code
9	License server version

GetLicenseStats

No arguments

Return type: string array

Returns information about the license checkout process. This function is typically used to provide diagnostic information when a license checkout fails.

Return value

Returns an array of strings. Each entry provides details of each license location. The first entry is always the license path for license files. This is always the License directory under the SIMetrix root. Subsequent entries refer to network license servers and there could be more than one of these.

Each entry is a semi-colon delimited list of values in the form: *location;type;checkout successful;checkout time;error code*. *type* may be 'path' or 'server'. *error code* will be 0 if successful otherwise it will be a negative number according to the cause of failure. A list of error codes is provided in the FLEXIm end user documentation provided on the install CD. *checkout time* is the time taken to check out the license.

GetLine

Type	real
Description	file handle
Compulsory	Yes
Default	

Return type: string

Returns a single line from a file.

Argument 1

Handle as returned by the [OpenFile \(page 247\)](#) function

Return Value

The first call to this function after opening the file, will return the first line in the file. Subsequent calls will return the remaining lines in sequence. The function will return an empty vector when there are no more lines in the file. The function will also return an empty vector if the file handle is not valid.

GetLongPathName

Type	string
Description	Path
Compulsory	Yes
Default	

Return type: string array

Returns long path name for path specified either as a long or short path. Long and short paths are only supported in Windows. On Linux systems, this function simply returns the input argument.

Argument 1

Input path. Maybe full or partial and the function will return its argument in the same form. (That it, it won't convert to a full path). If the input path does not exist, this function will simply return its argument unmodified.

See also: [GetShortPathName](#)

GetMenuItems

Type	string	string
Description	Menu path	Options
Compulsory	Yes	No
Default		

Return type: string array

Returns all menu item names in the specified menu.

Argument 1

Specifies the path for the menu as it would be provided to the DefMenu command (page 368) but without the menu item name. For example, the command to define the command shell's New Schematic menu is similar to:

```
DefMenu "Shell|&File|&New Schematic" "NewSchem /ne"
```

Shell|&File is the menu path and this what the GetMenuItems function expects.

Argument 2

Can be set to 'recurse'. This instructs the function to recurse into sub-menus and list all menu definitions. The definitions are given as semi-colon delimited strings providing the menu accelerator (if present), a unique ID and the full path of the menu.

Return Value

Returns a string array listing all the menu item names. E.g.

```
GetMenuItems('Shell|&File')
```

returns all the menu items in the command shell's File menu.

GetModelFiles

No arguments

Return type: string array

Returns a list of currently installed device models.

GetModelName

Type	string
Description	Instance name
Compulsory	Yes
Default	

Return type: string

Returns the model name used by an instance. The model name is the name for the parameter set (e.g. 'QN2222') as opposed to 'model type name' (e.g. 'npn') and 'internal device name' (e.g. 'BJT').

Note that all simulator devices use a model even if it is not possible for the device to use a .MODEL statement. Inductors, for example, are not permitted a .MODEL control but they nevertheless all refer to an internal model which is always called '\$Inductor'.

GetModelParameterNames

Type	string
Description	Internal device name
Compulsory	Yes
Default	

Return type: string array

Returns the names or default values of all real valued parameters for a device model.

Argument 1

Internal device name. This is returned by [GetInternalDeviceName \(page 171\)](#) and [GetModelType \(page 179\)](#)

Argument 2 - UNSUPPORTED

If a second argument is supplied set to 'default', the function will instead return the default values used for the device's parameter names. This doesn't work correctly for all simulator devices and so is currently unsupported.

GetModelParameterValues

Type	string	string
Description	Model name	Parameter name
Compulsory	Yes	No
Default		All values returned if omitted

Return type: string array

Returns the values of all parameters of the specified model. (Defined by 'model name' e.g. 'Q2N2222'). This function reads the values from the simulator and requires that a simulation has been run or checked. The returned array with arg2 omitted is of the same size as the array returned by [GetModelParameterNames \(page 178\)](#) for the same device and the values and parameter names map directly.

Argument 1

Model name. (Model name is the user name for a model parameter set as defined in the .MODEL control e.g. 'Q2N2222').

Argument 2

Parameter name. If specified return value will be a single value for the specified parameter. If omitted, the values for all parameters will be returned.

GetModelType

Type	string
Description	Model name
Compulsory	Yes
Default	

Return type: string

Returns internal device name given user model name. The internal device name is a name used internally by the simulator and is required by some functions. See [“GetInternalDeviceName” on page 171](#) for full details. The user model name is the name of a model parameter set defined using .MODEL. E.g. 'Q2N2222'.

Important: this function only works for models used by the current simulation. That is, you must run or check a simulation on a netlist that uses the specified model before calling this function.

Argument 1

User model name. See above.

GetModifiedStatus

Type	real
Description	schematic id
Compulsory	No
Default	Current schematic

Return type: real

Returns whether the specified schematic has been modified.

Argument 1

Specifies the id as returned by the “OpenSchematic” function. If omitted the current schematic will be used

GetNamedSymbolPins

Type	string	string
Description	Symbol name or component path	options
Compulsory	Yes	No
Default		'symbol'

Return type: string array

Returns the names for all pins of the specified symbol or hierarchical component.

Argument 1

Internal symbol name. This is the name used internally to reference the symbol and should not be confused with the ‘user name’ which is usually displayed by the user interface.

The symbol must be present in a currently installed library. If argument 2 is set to ‘comp’ then this argument instead specifies the file system path name of a component (.SXCMP) file.

Argument 2

See above

Return Value

Returns a string array of length equal to the number of pins on the specified symbol. If the symbol or component cannot be found the function returns an empty vector ([page 28](#)).

GetNamedSymbolPropNames

Type	string	string
Description	Internal symbol name	options
Compulsory	Yes	No
Default		'symbol'

Return type: string array

Returns names of all properties defined for a library symbol.

Argument 1

Internal symbol name. This is the name used internally to reference the symbol and should not be confused with the 'user name' which is usually displayed by the user interface.

The symbol must be present in a currently installed library. If argument 2 is set to 'comp' then this argument instead specifies the file system path name of a component (.SXCMP) file

Argument 2

See above

Return Value

Returns a string array holding the names of all the symbol's properties. If the symbol or component cannot be found the function returns an empty vector ([page 28](#)).

GetNamedSymbolPropValue

Type	string	string	string
Description	Internal symbol name	Property name	options
Compulsory	Yes	Yes	No
Default			'symbol'

Return type: string

Returns the value of a property defined for a library symbol.

Argument 1

Internal symbol name. This is the name used internally to reference the symbol and should not be confused with the ‘user name’ which is usually displayed by the user interface.

The symbol must be present in a currently installed library. If argument 3 is set to ‘comp’ then this argument instead specifies the file system path name of a component (.SXCMP) file

Argument 2

Property name.

Argument 3

See above

Return Value

Returns a string holding the value of the selected property. If the symbol/component or property do not exist the function will return an empty vector ([page 28](#)).

GetNearestNet

No arguments

Return type: string array

Return Value

Returns a string array of length 3 providing information on the net nearest the mouse cursor. The elements of the array are defined in the following table:

Index	Description
0	Local net name e.g. V1_P.
1	Net name prefixed with hierarchical path e.g. U1.V1_P
2	‘1’ if the net is a bus connections, otherwise ‘0’

GetNonDefaultOptions

No arguments

Return type: string array

Returns names of all .OPTION settings in the most recent simulation that were not at their default value.

GetNumCurves

Type	string
Description	Curve id
Compulsory	Yes
Default	

Return type: real

Returns the number of curves in curve group. This is applicable to curves plotted for a Monte Carlo analysis.

GetOpenSchematics

Type	real
Description	User index of schematic window
Compulsory	No
Default	Currently selected window

Return type: string array

Returns the path names of all schematics in the specified window.

Argument 1

Index of schematic window. This is returned as the user index field by the function [GetWindowNames \(page 208\)](#) with the full option specified. If omitted, the function will operate on the currently selected schematic window.

Return value

A string array containing the full path names all schematics in the specified window.

GetOption

Type	string
Description	option name
Compulsory	Yes
Default	

Return type: string

Returns the value of the *option variable* of name given as argument. *Option variables* are created using the Set command - see the “Sundry Topics” chapter of the *User’s Manual* for details on *option variables*. The GetOption function returns FALSE if the option does not exist and TRUE if it exists but has no value.

GetPath

Type	string
Description	Item name
Compulsory	Yes
Default	

Return type: real

Returns full path name of one of the following.

Argument value	Function
ScriptDir	Script directory
StartUpDir	Start up directory
StartUpFile	Start up script
BiScriptDir	Built-in script directory
ExeDir	Directory containing executable file.
TempDataDir	Temporary simulation data directory
DocsDir	Windows: File system directory for the “My Documents” folder Linux: \$HOME
ShareDir	The directory where the directories for symbol and model sub-directories are expected to reside. Typically Program Files\SIMetrix50\support on Windows and /usr/local/simetrix_50/share on Linux.

GetPlatformFeatures

No arguments

Return type: string array

Returns information on availability of certain features that are platform dependent.

Return value

Currently a string of length 4 defined as follows

Index	Description
0	Is 'ShellExecute' function implemented. 'true' or 'false'
1	Obsolete
2	Is 'VersionStamp' function implemented. 'true' or 'false'
3	Is context sensitive help implemented. 'true' or 'false'

GetPrinterInfo

No arguments

Return type: string array

Returns array of strings providing system printer names and current application default printer. Under Linux this will only return valid information if CUPS (Common Unix Printing System) is installed and running. If CUPS is not running then the return value for index 0 will always be -1. It will also return -1 if there are no printers installed even if CUPS is running. Format is as follows:

Index	Description
0	Number of printers available in system
1	Index of printer that is currently set as default. (This is the default for the application <i>not</i> the system default printer - see below)
2 onwards	List of printer names.

Example

The following is an example of executing the command "Show GetPrinterInfo"

```

Index  GetPrinterInfo()
0      '4'
1      '2'
2      'HP LaserJet 4L to file'
3      'HP LaserJet 4L'
4      'Acrobat PDFWriter'
5      'Acrobat Distiller'

```

The default index is 2 so this means that 'Acrobat PDFWriter' is currently set as the default printer. This is the current default for the *application* and is what will be set when you open a Print dialog box. When SIMetrix starts, it will be initialised to the

system default printer but changes whenever you select a different printer in any of the printer dialogs.

GetPrintValues

No arguments

Return type: string array

Returns the names of all quantities specified in .PRINT controls in the most recent simulation run.

GetReadOnlyStatus

Type	real
Description	schematic id
Compulsory	No
Default	Current schematic

Return type: real

Returns the read only status of the specified schematic.

Argument 1

Specifies the id as returned by the “[OpenSchematic](#)” function. If omitted the current schematic will be used.

Return Value

Returns 1.0 if the schematic is read-only. Otherwise returns 0.0

GetSchematicVersion

No arguments

Returns version information for the currently selected schematic.

Return value

Returns an array of length 3 with each element defined in the following table.

Index	Description
0	Format version. This will be an integer defining the format of the schematic binary file. Possible values and the SIMetrix versions for which those formats were used are: 102 Version 1.0 to 2.02 250 Version 2.5 to 4.0 420 Version 4.1 421 Version 4.2 422 Version 4.5 423 Version 5.0 - 5.2 424 Version 5.3 425 Version 5.4 426 Version 5.5 0 ASCII schematic
1	User version. Each time the schematic is saved this value is incremented
2	Exact version of SIMetrix that was used to save the file. Only valid if saved with version 5.4 or later. Otherwise this field will be empty. Version includes the maintenance suffix letter. E.g. 5.4e

GetSchemTitle

No arguments

Return type: string

Returns the title of the current schematic.

GetSelectedCurves

No arguments

Return type: string array

Returns array of curve id's for selected curves.

GetSelectedGraphAnno

No arguments

Return type:string

Returns the ID for the currently selected graph annotation object. If no object is selected, the function returns '-1'. If no graphs are open, the function returns an empty vector.

See “[Graph Objects](#)” on [page 448](#) for information on graph annotation objects.

GetSelectedYAxis

No arguments

Return type:string

Returns id of selected y-axis.

GetShortPathName

Type	string
Description	Path
Compulsory	Yes
Default	

Return type:string

Returns short path name for path specified either as a long or short path. Long and short paths are only supported in Windows. On Linux systems, this function simply returns the input argument.

Argument 1

Input path. Maybe full or partial and the function will return its argument in the same form. (That it, it won't convert to a full path). If the input path does not exist, this function will simply return its argument unmodified.

See also: [GetLongPathName](#)

GetSimConfigLoc

No arguments

Return type:string

Returns the location of the simulator's configuration information. This function returns its result in an identical form to the [GetConfigLoc \(page 148\)](#) function.

GetSimetrixFile

Type	string	string array	string
Description	file type	options	initial file
Compulsory	Yes	No	No
Default		<<empty>>	<<empty>>

Return type: string

Function opens a dialog box to allow the user to select a file. Returns the full path name to the selected file or an empty string if cancelled.

Argument 1

String to define one of the standard SIMetrix file types. This determines the files that will be displayed. Possible values are:

'Schematic'	Schematic files
'Data'	Data files
'Text'	Text files
'LogicDef'	Logic definition files as used by the arbitrary logic block
'Script'	Script files
'Model'	Model files
'Catalog'	Catalog files
'Graph'	Graph files
'Component'	Schematic component files
'Symbol'	Symbol library file

The type selected determines the files to be displayed controlled by their extension. The extension associated with each file type can be set with the options dialog box opened by menu File|Options|General... .

Argument 2

String array that specifies a number of options. Any or all of the following may be included:

'ChangeDir'	If present, the current working directory will change to that containing the file selected by the user
'Open'	If present a "File Open" box will be displayed other wise a "Save As" box will be displayed.
'NotExist'	If used with 'Open', the file is not required to already exist to be accepted
'All'	If present an "All files" entry will be added to the "Files of type" list

Argument 3

Initial file selection.

GetSimplisAbortStatus

No arguments

Return type:string

Determines whether a request to abort a SIMPLIS run has been received.

Return value

Returns 'AbortRequested' if a SIMPLIS abort request has been received otherwise returns 'NoAbort'.

GetSimplisExitCode

No arguments

Return type:real

Returns the application exit code for the most recent SIMPLIS run. This may be used to determine whether SIMPLIS completed its run successfully.

Return value

Returns a single value according to the most recent SIMPLIS run.

GetSimulationErrors

No arguments

Return type: String array

Returns all errors raised by the most recent simulation

Return value

Returns a string array with all errors raised by the most recent simulation. If the simulation ran correctly with no errors, an empty value (see [page 28](#)) will be returned. Note that this function only returns error messages; it does not return warnings.

GetSimulationInfo

No arguments

Return type:string array

Returns a string array of length 11 providing the following information about the most recent simulation:

Index	Description
0	Netlist path
1	List file path
2	Using data file 'True' or 'False'
3	Name of user specified data file
4	Collection name (obsolete)
5	.OPTIONS line specified at RUN command
6	Analysis line specified at RUN command
7	Reserved for future use
8	Reserved for future use
9	Reserved for future use
10	Reserved for future use

GetSimulationSeeds

No arguments

Return type:real array

Returns the seeds used for the most recent run. If this run was a Monte Carlo analysis, the return value will be an array of length equal to the number of Monte Carlo steps. Each element will hold the seed used for the corresponding step.

GetSimulatorEvents

***** UNSUPPORTED FUNCTION ***** see [page 72](#)

No arguments

Return type: string array

This function was developed to aid simulator development and also to assist identifying causes of convergence failure. It has also been used to detect the success or otherwise of a simulation run called by a script by examining the last event in the return value.

The following is accurate for version 4.0b. Later versions may be different but any changes are likely to be made by adding additional events or/and adding additional fields to the event line.

Returns a string array, each element of which describes an event that occurred during the most recent simulation. Each element is a string consisting of a number of values separated by semi-colons. The first value is the name of the event. This can be one of the following:

Singular matrix	Singular matrix - may lead to abort but not necessarily.
Floating point error	Floating point error occurred such as divide by zero or log of a negative number. May lead to abort but depends on where it occurred
Operating point complete	
Operating point failed	
GMIN step started	
Source step started	
Pseudo transient started	
Job started	Always the first event
Job complete	Final event
Job failed	Final event
Job paused	Final event
Job resumed	
Job aborted	Final event
Node limit exceeded	Means that a node voltage exceeded the value of the NODELIMIT option. (Default 1e50). The iteration is rejected when this happens but does not directly lead to an abort.
Iteration succeeded (full)	

Iteration failed (full)	
Load failed	Iteration failed because device equations could not be evaluated. Usually caused by excessive junction voltage
LTE reject (full)	Time step rejected because local truncation error too high
LTE accept (full)	Local truncation error below tolerance. Time step accepted

The items marked “(full)” will only be listed if the .OPTIONS setting FULLEVENTREPORT is specified when the simulator is run.

The remaining values are listed below:

Field number	Description
1	Top level analysis mode. One of: 'none', 'Op', 'Tran', 'AC', 'Sweep', 'Noise', 'TF', 'Sensitivity', 'Pole-zero'
2	Operating point mode. One of: 'none', 'JI2', 'GMIN', 'Source', 'PTA'
3	Transient analysis time
4	Time step
5	Real time measured from start of run (not output for all events)
6	Iteration number
7	Event specific message

GetSimulatorMode

No Arguments

Return type: string

Returns the simulator mode of the current schematic. Return value may be 'SIMetrix' or 'SIMPLIS'.

GetSimulatorOption

Type	string
Description	Option name
Compulsory	Yes
Default	

Return type: string

Returns the value of a simulator option as used by the most recent analysis. The argument may be any one of the option names defined for the .OPTIONS control. E.g.

```
GetSimulatorOption('RELTOL')
```

will return the value of RELTOL for the most recent run. If the option value was not explicitly specified in a .OPTIONS control, its default value will be returned.

GetSimulatorStats

No arguments

Return type: real array

Returns a 26 element real array providing statistical information about the most recent run. The meaning of each field is described below:

Index	Description
0	Number of event driven outputs
1	Number of event driven ports
2	Number of event driven instances
3	Number of event driven nodes
4	Number of equations (= matrix dimension = total number of nodes including internal nodes)
5	Total number of iterations
6	Number of transient iterations
7	Number of JI2 iterations. (First attempt at DC bias point)
8	Number of GMIN iterations
9	Number of source stepping iterations
10	Number of pseudo transient analysis iterations
11	Number of time points
12	Number of accepted time points

Index	Description
13	Total analysis time
14	Transient analysis time
15	Matrix load time. (The time needed to calculate the device equations)
16	Matrix reorder time.
17	Matrix decomposition time.
18	Matrix solve time
19	Size of state vector
20	Parameter evaluation time
21	Matrix decomposition time (transient only)
22	Matrix solve time (transient only)
23	Circuit temperature
24	Circuit nominal temperature
25	Number of matrix fill-ins
26	Simulator Initialisation time
27	Number of junction GMIN iterations
28	Time to process digital events
29	“Accept” time. This is the time used for processing transient time points after the simulator has accepted it. This includes the time taken to write out the data.

GetSimulatorStatus

No arguments

Return type: string

Returns the current status of the simulator. May be one of the following values:

Paused	Simulator paused
InProgress	Simulation in progress. (The only situation where this value can be returned is when calling this function remotely using the SxCommand utility with the -immediate switch. It isn't otherwise possible to call a function while a simulation is running.)
ConvergenceFail	Last simulation failed because of no convergence
SimErrors	Last simulation failed because of a run time error

NetlistErrors	Last simulation failed because of a netlist error
Warnings	Last simulation completed with warnings
Complete	Last simulation successful
None	No simulation has been run

GetSoaDefinitions

No arguments

Return type: string array

Returns all Safe Operating Area definitions specified in the most recent analysis.

Return Value

Returns an array of strings with each string in the form:

label;minvalue;maxvalue;xwindow;derating;type

Where:

<i>label</i>	The label specification on the .SETSOA line
<i>minvalue</i>	Minimum value
<i>maxvalue</i>	Maximum value
<i>xwindow</i>	Window width - the time the limits must be exceeded for the violation to be recorded
<i>derating</i>	Derating factor
<i>type</i>	'Peak' or 'Mean'

GetSoaMaxMinResults

No arguments

Return type: string array

Returns the maximum and minimum values reached for all SOA definitions.

Return Value

Returns an array of strings defining max and min values reached. Each element in the array corresponds to the elements returned by the [GetSoaDefinitions](#) function. Each string is of the form:

min_val;min_reached_at;max_val;max_reached_at;max_mean

Where

<i>min_val</i>	Minimum value reached
<i>min_reached_at</i>	Time at which the minimum value was reached
<i>max_val</i>	Maximum value reached

<i>max_reached_at</i>	Time at which the maximum value was reached
<i>max_mean</i>	Maximum mean value

Notes

This function returns the maximum and minimum values returned for all SOA definitions regardless of whether or not the limits were violated

GetSoaOverloadResults

Type	string
Description	options
Compulsory	No
Default	

Return type: real array

Returns the overload factor for each SOA definition.

Argument 1

String array consisting of one or both of the values: 'ignorewindow' or 'derated'. If 'ignorewindow' is specified, then the function will not return data for SOA specifications that include a window. If 'derated' is included, the values returned allow for any derating factor. For example, if the limit is 40V with 80% derating and the maximum value reached was 38V, the overload factor with 'derated' specified will be $38/(40*0.8) = 1.1875$. Without 'derated' specified, the overload factor would be $38/40 = 0.95$

Return Value

Returns an array of reals defining the overload factor for each SOA definition. Each element in the array corresponds to the elements returned by the [GetSoaDefinitions](#) function.

GetSoaResults

No arguments

Return type: string

Returns the SOA results for the most recent simulation.

Return Value

Returns an array of strings, each one describing a single SOA failure. Each string is a semi-colon delimited list with fields defined below.

Field	Description
0	SOA Label
1	Start of failure
2	End of failure
3	'under' or 'over'. Defines whether the test fell below a minimum limit or exceeded a maximum limit.
4	Value of limit that was violated

GetSymbolArcInfo

No arguments

Return type: real array

Returns an array of length 4 providing information on selected arcs/circles/ellipses in the symbol editor. Format is as follows:

Index	Description
0	Swept angle in degrees
1	Height/Width
2	Number of selected arcs/circles/ellipses
3	0 if all selected arcs/circles/ellipses are identical to each other. Otherwise 1

GetSymbolFiles

No arguments

Return type: string array

Returns full paths of all installed symbol library files.

GetSymbolInfo

No arguments

Return type: string array

Returns a string array of length 3 providing information on the symbol in the currently selected symbol editor sheet. If no symbol editor sheet is open the function returns an empty vector.

Format of the return value is:

Index	Description
0	Symbol name
1	Symbol description
2	Symbol catalog
3	Path to symbol library or component file where the symbol definition is located. If the symbol is not found in any symbol library, this element will be empty.
4	Type of symbol. One of two values: 'Symbol' Regular symbol stored in a library 'Component' Hierarchical component
5	Flags. Currently values can only be 0 or 1. Future versions may use additional bits. For forward compatibility, test this value using the Field function (page 135) to test bit 0. The value reports the state of the 'All references to symbol automatically updated' check box when the symbol was saved. If checked, this value will be 1 otherwise 0.

GetSymbolOrigin

No arguments

Return type: real array

Returns the location of the origin point of the symbol currently open in the symbol editor. The origin is the location of the point 0,0 on the symbol. It is in turn located at a position relative to the *reference point*. The reference point is an absolute location defined by the symbol's geometry. If the symbol has pins, it is the top left of a rectangle that encloses all the pins. Otherwise it is the top left of a rectangle that encloses all the segments.

Return Value

2 element real array. Index 0 is the x-coordinate while index 1 is the y-coordinate. The units are 100/grid square.

See Also

[SetOrigin \(page 422\)](#)

GetSymbolPropertyInfo

Type	string
Description	property name
Compulsory	No
Default	

Return type: string array

Returns a string array of length 5 providing information on either a single property as defined in the argument or the currently selected properties. Format of result is as follows:

Index	Description
0	Property name
1	Property flags value (see "Attribute flags" on page 404 for details.)
2	Property value
3	Number of properties selected
4	Number of pins selected

If more than one property or pin is selected, the information provided in 0-2 above will be on one of them but there are no rules to determine which. The displayed names used for pins are represented as properties and this function can be used to gain information about them. The equivalent property name for a pin is the pin name prefixed with \$Pin\$.

GetSymbolPropertyNames

No arguments

Return type: string array

Returns string array containing names of all selected properties in the currently open symbol editor sheet. If there are no selected properties or the symbol editor is not open, the function will return an empty vector. Note the displayed names used for pins are represented as properties and this function can be used to list them. The equivalent property name for a pin is the pin name prefixed with "\$Pin\$".

GetSymbols

Type	string	string
Description	options	catalog name
Compulsory	No	No
Default	'name'	'all'

Return type: string array

Returns a string array containing information about installed symbols.

Argument 1

Defines what the function returns as defined in the following table

Option value	
'description'	Returns the user name of each symbol
'catalogs'	Returns the catalog names for each of the symbols. The catalog defines how the symbol user name is displayed in the symbol dialog display as opened by the SelectSymbolDialog (page 288) . It consists of one or more strings separated by semi-colons. Each string defines a node in the tree list display
'tree'	'catalogs' and 'description' merged together but separated by a semi-colon
"	Internal symbol name

For example, the standard three terminal NPN symbol has an internal name of 'npn', a catalog of 'Semiconductors;BJTs' and a description of 'NPN 3 Terminal'. The value returned by the 'tree' option would be 'Semiconductors;BJTs;NPN 3 Terminal'

Argument 2

Specifies a filter that selects symbols according to catalog. May be prefixed with '-' in which case all symbol *not* belonging to the specified catalog will be returned.

Return Value

Returns string array providing the symbol info as defined by arg 1 and 2. If there are no symbol libraries installed or there are no symbols with the specified catalog, an empty vector will be returned. (See [page 28](#))

GetSymbolText

Type	string	string
Description	Symbol name. If 'all' text for all symbols will be output.	local global
Compulsory	Yes	No
Default		'global'

Return type: string array

Returns definitions for specified symbols in 'Compact Text Format' (see below). Currently this function only returns definitions of generated symbols. These are symbols created either with the symbol generator or as text files using PinDef (page 397) commands.

If argument 2 = 'schematic' symbols used in the current schematic will be returned. Otherwise symbols in the global library will be returned.

Compact Text Format

This format is also used by the SymbolGen function (page 301). This consists of an array of strings each containing semi-colon delimited values. Each symbol is of the following form

Header
Pin definitions
Property definitions

Header:
symbol name;number of pins;number of properties;symbol description;rectangle|triangle

Pin definition:
pin name;pin number;pin position;pin visibility
pin position left|right|top|bottom
pin visibility vis|novis

These values map exactly to the PinDef (page 397) command arguments

Property definition:
property name;initial value;flags
These values map exactly to the AddProp (page 344) command arguments

GetTimerInfo

Type	real
Description	Timer ID
Compulsory	Yes
Default	

Return type: string array

Returns information about a timer.

Argument 1

Timer ID as returned by [CreateTimer \(page 93\)](#)

Return Value

Returns a string array of length 2. The first element is the command called by the timer, the second element is the current interval. A value of zero means that the timer is currently stopped.

GetSystemInfo

No arguments

Return type: string array

Returns information about the user's system

Return Value

String array of length 7 as defined by the following table:

Index	Description
0	Computer name
1	User log in name
2	Returns 'Admin' if logged in with administrator privilege otherwise returns 'User'. On Linux this returns 'Admin' if the program was started as 'root'.

Index	Description
3	Available system RAM in bytes
4	Operating system class. Returns either 'WINNT' or 'UNIX'. With earlier versions of SIMetrix this maybe WIN9X if running on Windows 95, 98 or ME. These platforms are not supported by version 5.
5	Operating System description. Returns descriptive name for operating system. For Linux versions, always returns 'GNU/Linux'
6	Linux only. Returns display name. E.g. ':0.0' for the primary display. Empty for Windows versions

GetToolButtons

Type	string
Description	Button class
Compulsory	No
Default	All

Return type: string array

Returns name and description for available tool buttons.

Argument 1

Class name of buttons. With no user defined buttons, this can be empty or 'component'. If 'component' only buttons intended for placing schematic symbols will be returned. Otherwise all buttons available will be returned.

If user defined buttons have been created using the [CreateToolButton \(page 356\)](#) command, this argument may be set to any value used for the /class switch in which case only buttons defined with that /class switch value will be returned.

Return Value

String array of button specifications. Each entry contains two values separated by a semi-colon. The first value is the name of the button as can be used to add buttons to a toolbar using the [DefineToolBar \(page 362\)](#) command. The second value is a description of the button.

See Also

["CreateToolBar" on page 355](#)

["DefButton" on page 361](#)

["SetToolBarVisibility" on page 423](#)

GetUncPath

Type	string
Description	path
Compulsory	Yes
Default	

Return type: string

Returns the given path in UNC form, i.e: `\\servername\servershare\server-localpath`. This function's main purpose is to convert windows drive letters to a consistent format.

Argument 1

Path of file in any form. Typically this would include a drive letter on windows.

Return value

Path in UNC form. Note that if a drive letter on a local machine is used in the path, this function will return the original path unmodified even if a network share is defined for that drive.

Linux does not support the UNC format and instead this function, it will convert sym-links to real paths.

GetUserFile

Type	string	string	string array	string
Description	file filter	default extension	options	initial file
Compulsory	No	No	No	No
Default			<<empty>>	<<empty>>

Return type: string

Function opens a dialog box to allow the user to select a file.

Argument 1

Defines file filters. The 'save as type' list box may contain any number of entries that defines the type of file to be displayed. This argument defines the entries in this list box.

Each entry consists of a description followed by a pipe symbol (|) then a list of file extensions separated by semi-colons (;). Entries are also separated by the pipe (|) symbol. For example: to list just schematic files enter:

```
“Schematic files|*.xsch;*.sch”
```

Note that the text is enclosed in both single and double quotes. Strings in expressions are denoted by single quotes as usual but the semi-colon is normally used to separate commands on a single line. This is inhibited by enclosing the whole string in double quotes.

If you wanted to provide entries for selecting - say - both schematics and netlists, you could use the following:

```
“Schematic files|*.xsch;*.sch|Netlist files|*.net;*.cir”
```

Argument 2

The default extension specified without the dot. This is the extension that will automatically be added to the file name if it does not already have one of the extensions specified in the filter.

Argument 3

String array that specifies a number of options. Any or all of the following may be included:

'ChangeDir'	If present, the current working directory will change to that containing the file selected by the user
'Open'	If present a File Open box will be displayed other wise a Save As box will be displayed.
'NotExist'	If used with 'Open', the file is not required to already exist to be accepted
'ShowReadOnly'	If present and 'Open' is also specified, an Open as read-only check box will be displayed. The user selection of this check box will be returned in either the second or third field of the return value.
'FilterIndex'	If specified, the type of file selected by the user will be returned as an index into the list of file filters specified in argument 1. So, 0 for the first, 1 for the second etc.

Argument 4

Initial file selection.

Return value

String array of length between 1 and 3 as described in the following table:

Option 'ShowReadOnly'	Option 'FilterIndex'	Return value
No	No	Path name only
Yes	No	2 element array: index=0 path name index=1 Read only checked - 'TRUE' or 'FALSE'
No	Yes	2 element array index=0 path name index=1 Filter index selected
Yes	Yes	3 element array index=0 path name index=1 Filter index selected index=2 Read only checked - 'TRUE' or 'FALSE'

GetVecStepParameter

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: string

This function retrieves the name of the parameter that was stepped to obtain the vector data supplied. It will only return a meaningful result for data vectors generated by a multi-step analysis. For example, if an analysis was performed which stepped the value of the resistor R7, this function would return 'R7' when applied to any of the data vectors created by the simulator. If the analysis was a Monte Carlo run, the function will return 'Run'.

If this function is applied to single division data as returned by a normal single step run, the return value will be an empty vector.

GetVecStepVals

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real array

This function retrieves the values assigned to the parameter that was stepped to obtain the vector data supplied. It will only return a meaningful result for data vectors generated by a multi-step analysis. For example, if an analysis was performed which stepped the value of the resistor R7 from 100Ω to 500Ω in 100Ω steps, this function would return [100, 200, 300, 400, 500]. If the analysis was a Monte Carlo run, the function will return the run numbers starting from 1.

If this function is applied to single division data as returned by a normal single step run, the return value will be an empty vector.

GetWindowNames

Type	string
Description	Options
Compulsory	No
Default	

Return type: string array

Returns names of current windows. Result can be supplied as argument to Focus command using /named switch or /userid switch.

Argument 1

If set to 'full', this function will return more detailed window information in the form of a semi-colon delimited string with the following three fields:

Index	Description
0	Window type. One of, 'Shell', 'Schematic', 'Graph' or 'Symbol'
1	User index. Integer that can supply to Focus (page 380) command using /userid switch, the GetGraphTabs (page 166) function and GetOpenSchematics (page 183) function. Note that the command shell always has a user index of -1
2	Window title

GetXAxis

No arguments

Return type: string

Returns the id of the x-axis in the currently selected graph.

GraphLimits

No arguments

Return type: real array

The x and y axis limits of the currently selected graph and axis type (log/linear). Function will fail if there are no selected graphs. Meaning of each index of the 6 element array are as follows:

Index	Description
0	x-axis lower limit
1	x-axis upper limit
2	y-axis lower limit
3	y-axis upper limit
4	1 if x-axis is logarithmic, 0 if linear
5	1 if y-axis is logarithmic, 0 if linear

GroupDelay

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real array

Returns the group delay of the argument. Group delay is defined as:

$$\frac{d}{dx}(\text{phase}(y)) \cdot \frac{1}{2\pi}$$

where y is the supplied vector and x is its reference. The GroupDelay function expects the result of an AC analysis where y is a voltage or current and its reference is frequency.

This function will yield an error if its argument is complex and has no reference.

Groups

Type	string
Description	Title Name
Compulsory	No
Default	'name'

Return type: string array

Returns names of available groups. The first element (with index 0) is the current group. If the argument 'Title' is provided, the full title of the group is returned. More information about groups can be found in [“Groups” on page 37](#).

Hash

Type	string
Description	
Compulsory	Yes
Default	

Return type: string

Returns a 'hash' value for the supplied string. A hash value is an integer value similar to a check sum.

HasLogSpacing

Type	real
Description	Vector
Compulsory	Yes
Default	

Return type: real

Performs a simple test to determine whether the supplied vector is logarithmically spaced. The return value is 1.0 if the vector is logarithmically spaced and 0.0 otherwise. Note the function expects to be supplied with x-values.

HasProperty

Type	string	string	string	real
Description	Property name	property name to identify	property value to identify	Schematic handle
Compulsory	Yes	No	No	No
Default				-1

Return type: real

Determines whether a particular instance possesses a specified property.

Argument 1

Property name.

Argument 2, 3

If present, these arguments identify the instance to be tested for property ownership. If neither of these arguments are present the currently selected instance will be tested. If only argument 2 is present, any instance possessing the property it specifies will be tested.

If more than instance is identified one of them will be tested but there are no rules to determine which.

Argument 4

Schematic handle as returned by the OpenSchematic function (page 249). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

Return value

Outcome of test: TRUE (1) or FALSE (0). If no instance matches argument 2 and 3, an empty value (page 28) will be returned.

HaveFeature

Type	string
Description	Feature name
Compulsory	Yes
Default	

Return type: real

Determines whether a specified license feature is available

Argument 1

Name of license feature. Currently may be one of, 'basic', 'advanced', 'micron', 'rtn', 'simplis_if', 'AD', 'schematic' or 'scripts'.

Return value

Returns 1.0 if the license feature is available otherwise it returns 0.0.

HaveInternalClipboardData

Type	string
Description	Data type
Compulsory	Yes
Default	

Return type: real scalar

Returns the number of items in the specified internal clipboard. The internal clipboard is currently only used for graph curve data.

Argument 1

The name of the internal clipboard to be queried. Currently there is only one internal clipboard so this argument must always be 'GraphCurve'.

Notes

Use [CurveEditCopy \(page 361\)](#) to copy graph curve data to the internal clipboard. Use the “Curve /icb *curve_index*” to plot a curve that resides in the internal clipboard.

HierarchyHighlighting

This function is used by the hierarchical highlighting system and its operation and argument list may be subject to change. Consequently, this function is not yet fully supported.

HighlightedNets

Type	real
Description	Schematic handle
Compulsory	No
Default	-1 (current schematic)

Return type: string array

Returns names for any wholly highlighted net names on the specified schematic.

Argument 1

Schematic handle as returned by the [OpenSchematic \(page 249\)](#) function. If omitted, the currently selected schematic will be used

Return Value

Returns the highlighted netnames as an array of strings.

Histogram

Type	real array	real	string
Description	Vector	Number of bins	Options
Compulsory	Yes	Yes	
Default			

Return type: real array

Creates a histogram of argument 1 with the number of bins specified by argument 2. The bins are divided evenly between the maximum and minimum values in the argument.

Argument 1

Vector to be processed

Argument 2

Number of bins

Argument 3

Set to 'step' to force output in a stepped style similar to a bar-graph.

Notes

Histograms are useful for finding information about waveforms that are difficult to determine by other means. They are particularly useful for finding "flat" areas such as the flat tops of pulses as these appear as well defined peaks. The Histogram() function is used in the rise and fall time scripts for this purpose.

Users should note that using this function applied to raw transient analysis data will produce misleading results as the values are unevenly spaced. If you apply this function to simulation data, you must either specify that the simulator outputs at fixed intervals (select the Output at .PRINT step option in the Simulator|Choose Analysis... dialog box) or you must interpolate the results using the Interp function - see [page 223](#)

Iff

Type	real array	real array	real array
Description	Test value	Result if test true	Result if test false
Compulsory	Yes	Yes	Yes
Default			

Return type: Same as args 2 and 3

If the first argument evaluates to TRUE (i.e. non-zero) the function will return the value of argument 2. Otherwise it will return the value of argument 3. Note that the type of arguments 2 and 3 must both be the same. No implicit type conversion will be performed on these arguments.

IIR

Type	real array	real array	real array
Description	Vector to be filtered	Coefficients	Initial conditions
Compulsory	Yes	Yes	No
Default			zero

Return type: real array

Performs Infinite Impulse Response digital filtering on supplied vector. This function performs the operation:

$$y_n = x_n \cdot c_0 + y_{n-1} \cdot c_1 + y_{n-2} \cdot c_2 \dots$$

Where:

x is the input vector (argument 1)

c is the coefficient vector (argument 2)

y is the result (returned value)

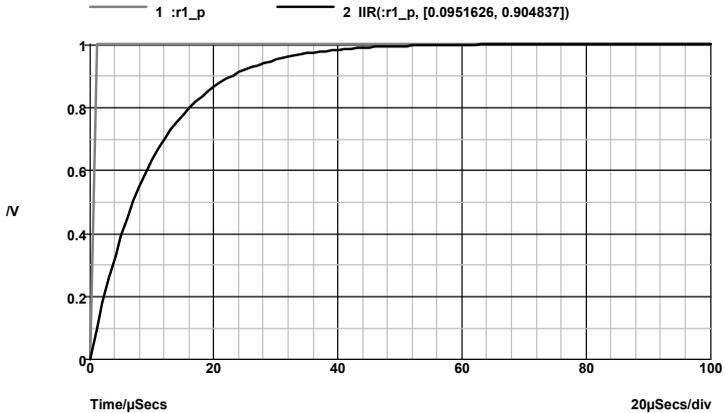
The third argument provides the “history” of y i.e. y_{-1} , y_{-2} etc. as required.

The operation of this function (and also the FIR function) is simple but its application can be the subject of several volumes! In principle an almost unlimited range of IIR filtering operations may be performed using this function. Any text on Digital Signal Processing will provide further details.

User's should note that using this function applied to raw transient analysis data will not produce meaningful results as the values are unevenly spaced. If you apply this function to simulation data, you must either specify that the simulator outputs at fixed intervals (select the Output at .PRINT step option in the Simulator|Choose Analysis... dialog box) or you must interpolate the results using the Interp function - see [page 223](#).

Example

The following graph shows the result of applying a simple first order IIR filter to a step:



The coefficients used give a time constant of 10 * the sample interval. In the above the sample interval was 1µSec so giving a 10µSec time constant. As can be seen a first order IIR filter has exactly the same response as an single pole RC network. A general first order function is:

$$y_n = x_n \cdot c_0 + y_{n-1} \cdot c_1$$

where $c_0=1-\exp(-T/\tau)$
 and $c_1=\exp(-T/\tau)$
 and τ =time constant
 and T =sample interval

The above example is simple but it is possible to construct much more complex filters using this function. While it is also possible to place analog representations on the circuit being simulated, use of the IIR function permits viewing of filtered waveforms after a simulation run has completed. This is especially useful if the run took a long time to complete.

im

Type	real/complex array
Description	
Compulsory	Yes
Default	

Return type: real array

Returns imaginary part of argument.

imag

Identical to **im()**

InputGraph

Type	string	string
Description	initial text	message
Compulsory	No	No
Default		<<empty>>

Return type: string

Opens a simple dialog box prompting the user for input. Dialog box position is chosen to keep selected graph visible if possible. Argument provides initial text, return value is text entered by user.

The function returns an empty vector if the user cancels the dialog box.

InputSchem

Type	string	string
Description	initial text	message
Compulsory	No	No
Default		<<empty>>

Return type: string

Opens a simple dialog box prompting the user for input. Dialog box position is chosen to keep selected schematic visible if possible. Argument provides initial text, return value is text entered by user.

The function returns an empty vector if the user cancels the dialog box.

Instances

Type	string	real
Description	Property name	Schematic handle
Compulsory	Yes	No
Default		-1

Return type: string array

Returns array of property values of property name specified as argument. A value will be returned for every instance on the schematic that possesses that property. (An instance is a schematic item represented by a symbol - components, ground symbols etc.) For example, Instances('ref') would return every component reference in the schematic.

Note that every instance has a unique 'Handle' property which is automatically assigned. This makes it possible to access every instance on the schematic.

The second argument is a schematic handle as returned by the OpenSchematic function (page 249). This allows this function to be used with a schematic that is not open or not currently selected. If omitted or -1, the currently selected schematic will be used.

The function will return an empty vector if no schematic is open or argument 2 is invalid. An empty *string* will be returned if no instance possess the specified property. The latter behaviour is not always convenient but is retained for backward compatibility. The PropValues2 function (page 261), with appropriate arguments, will return an empty *vector* when there is no match, and thus easier to use in many cases.

InstNets

Type	string
Description	Options
Compulsory	No
Default	

Return type: string array

Returns an array of strings holding netnames for each pin of the selected schematic instance. Circuit must have been netlisted for the result of the function to be meaningful. This function is used by the power script to find the power dissipated in a device.

If argument 1 is set to 'flat' the resulting netnames will be stripped of hierarchical references.

The function will return with an error if no instances are selected or more than one instance is selected.

InstNets2

Type	real	string	string	string
Description	Schematic handle	Property name	Property value	Options
Compulsory	Yes	Yes	Yes	No
Default				

Return type: string array

Returns an array of strings holding the netnames of a schematic instance defined by arguments 1 to 3.

Argument 1

Schematic handle as returned by the OpenSchematic function ([page 249](#)). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

Arguments 2,3

Property name and value to identify instance. If these arguments are not provided, the selected instance, if any, will be used instead. If there are no selected instances or no instances that match the arguments, the function will return an empty vector. If the arguments identify more than one instance, the function will return information for one of them but there are no rules to define which one.

Argument 4

If set to 'full', the full hierarchical path of the net names will be supplied. Otherwise the local names will be returned.

InstPins

Type	string	string	real
Description	property name	property value	Schematic handle
Compulsory	No	No	No
Default			-1

Return type: string array

Returns an array of strings holding pin names for each pin of either the selected instance or an instance identified by one or both arguments.

Argument 1, 2

Property name and value to identify instance. If these arguments are not provided, the selected instance, if any, will be used instead. If there are no selected instances or no instances that match the arguments, the function will return an empty vector. If the arguments identify more than one instance, the function will return information for one of them but there are no rules to define which one.

Using the 'HANDLE' property and its value will guarantee uniqueness.

Argument 3

Schematic handle as returned by the OpenSchematic function ([page 249](#)). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

InstPoints

Type	string	string	real
Description	property name	property value	Schematic handle
Compulsory	No	No	No
Default			-1

Return type: real array

Returns an array of length 3 providing XY co-ordinates and orientation of an instance.

Arguments 1,2

Property name and value to identify instance. If these arguments are not provided, the selected instance, if any, will be used instead. If there are no selected instances or no instances that match the arguments, the function will return an empty vector. If the arguments identify more than one instance, the function will return information for one of them but there are no rules to define which one.

Using the 'HANDLE' property and its value will guarantee uniqueness.

Argument 3

Schematic handle as returned by the OpenSchematic function ([page 249](#)). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

Return Value

Returns real array of size 3 as defined by the following table:

Index	Description
0	X co-ordinate
1	Y co-ordinate
2	Orientation: 0 to 7

The co-ordinates are those of the point defined to be at 0,0 in the symbol definition. The scaling used is 120 points to one grid square. (Grid refers to snap grid. This is the same as the visible grid for magnifications of 0.83 and higher.). Co-ordinates are relative. For a new schematic the zero point is at the top left corner of the window but this can change. The orientation values are as follows:

Orientation value	Description
0	Normal (as symbol def)
1	90 deg. clockwise
2	180 deg.
3	270 deg clockwise
4	Mirrored through y-axis
5	Mirrored through y-axis + 90 deg clock.
6	Mirrored through y-axis + 180 deg.
7	Mirrored through y-axis + 270 deg clock.

Note: Mirror through x-axis is equivalent to mirror through y with 180 rotation.

The values returned by this function can be used with Inst command using the /loc switch.

If no instance is identified by arguments 1 and 2 an empty value ([page 28](#)) will be returned.

InstProps

Type	string	string	real
Description	Property name	Property value	Schematic handle
Compulsory	No	No	No
Default			-1

Return type: string array

Returns an array of strings holding the names of all properties of an instance. The PropValue (page 259) or PropValues2 function (page 261) can be used to find values of these properties.

Arguments 1,2

Property name and value to identify instance. If these arguments are not provided, the selected instance, if any, will be used instead. If there are no selected instances or no instances that match the arguments, the function will return an empty vector. If the arguments identify more than one instance, the function will return information for one of them but there are no rules to define which one.

Using the 'HANDLE' property and its value will guarantee uniqueness.

Argument 3

Schematic handle as returned by the OpenSchematic function (page 249). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

Return Value

Array of strings with property values. Returns empty value (page 28) if no match to property name and value is found. Also returns empty value if the schematic handle is invalid.

Integ

Type	real array
Description	vector
Compulsory	Yes
Default	

Return type: real array

Integrates the argument with respect to its reference. See “Vector References” on page 42 for details.

The function uses simple trapezoidal integration.

An error will occur if the argument supplied has no reference.

Interp

Type	real array	real	real	real
Description	Vector to be interpolated	Number of points	Interpolation order	include last point
Compulsory	Yes	Yes	No	No
Default			2	FALSE

Interpolates the data in argument 1 either to a fixed number of points or at a specified interval.

Argument 1

Vector to be interpolated. The data should have a reference (x-values, see “[Vector References](#)” on page 42) but this is not compulsory when interpolating using a fixed number of points as opposed to a fixed interval.

Argument 2

Either the number of points or the x interval depending on the mode. (See argument 4 below)

Argument 3

Interpolation order. This can be any integer 1 or greater but in practice there are seldom reasons to use values greater than 4. If interpolating a signal containing fast pulses, interpolation order should be set to 1.

Argument 4

Two element boolean array, that is its values should be either TRUE (1) or FALSE (0). The second element specifies the mode. If 0 (FALSE) then the function uses the fixed number of points mode and argument 2 provides the number of points. If 1 (TRUE) the mode is fixed interval mode and argument 2 specifies the interval. The first element is only used with fixed number of points mode. If TRUE the final point of the interpolated result will coincide with the final point of the input vector and the interval between points is $T/(N-1)$ where T is the interval of the whole input vector and N is the number of points. If FALSE the interval is T/N and the final point is at a location T/N before the final input point. The latter behaviour is compatible with earlier versions and is also what should be used if the function is interpolating data to be used by the FFT function.

Return value

Returns the interpolated data.

Notes

The Interp() function overcomes some of the problems caused by the fact that raw transient analysis results are unevenly spaced. It is used by the FFT plotting scripts to provide evenly spaced sample points for the FFT() function.

IsComplex

Type	any
Description	vector
Compulsory	Yes
Default	

Return type: real

Returns 1 (=TRUE) if the supplied argument is complex and 0 (=FALSE) if the argument is any other type.

IsComponent

Type	string	string
Description	Property name	Property value
Compulsory	Yes	Yes
Default		

Return type: real

Determines whether a schematic instance is a hierarchical component. Schematic instance is defined using a property name and value.

IsFullPath

Type	string
Description	path
Compulsory	Yes
Default	

Return type: real

Returns TRUE if the supplied path name is a full absolute path.

Argument 1

File system path name

Return Value

TRUE if arg is a full absolute path. FALSE if it is a relative path.

IsModelFile

Type	string	string
Description	Path of file	Option
Compulsory	Yes	No
Default		

Return type: real

Returns 1 if the specified file contains .MODEL, .SUBCKT or .ALIAS definitions. Otherwise returns 0. The function will unconditionally return 0 if the file has any of the following extensions:

.EXE, .COM, .BAT, .PIF, .CMD, .SCH, .SXSCH, .SXDAT, .SXGPH

This will be overridden if the second argument is set to 'AllExt'.

IsNum

Type	any
Description	vector
Compulsory	Yes
Default	

Return type: real

Returns 1 (=TRUE) if the supplied argument is numeric (real or complex) and 0 (=FALSE) if the argument is a string

IsOptionMigrateable

Type	string
Description	option name
Compulsory	Yes
Default	

Return type: real

Determines if an option variable may be migrated in a version upgrade.

Argument 1

Option name

Return Value

Return 1.0 if the option name is migrateable otherwise returns 0.0.

This function is used in the script that is run when SIMetrix is started for the first time. Certain option variables (defined using the [Set \(page 420\)](#) command) are marked internally as 'migrateable' meaning that their values are transferred to a new version installation if the user requests that configuration settings are to be migrated.

IsSameFile

Type	string	string
Description	Path of file 1	Path of file 2
Compulsory	Yes	Yes
Default		

Return type: real

Compares two paths and returns true (1) if they point to the same file. The function takes account of the fact that the two arguments might try to access the same file by different methods. For example, on Windows, one file might use a drive letter while the other might use a server path.

IsScript

Type	string
Description	script name
Compulsory	Yes
Default	

Return type: real

Function to determine whether the supplied script name can be located. Calling this script will fail if this function returns FALSE. Note that the function doesn't check the script itself. It only determines whether or not it exists.

Argument 1

Script name

Return Value

Returns TRUE if the supplied script name can be located in the standard script path.

IsStr

Type	any
Description	vector
Compulsory	Yes
Default	

Return type: real

Returns 1 (=TRUE) if the supplied argument is a string and 0 (=FALSE) if the argument is numeric (real or complex).

JoinStringArray

Type	string array	string array
Description	First array	Second array
Compulsory	Yes	Yes
Default		

Return type: string array

Concatenates two string arrays to return a single array.

Argument 1

First array

Argument 2

Second array

Return Value

Array of strings of length equal to the sum of the lengths of arguments 1 and 2. Contains arguments 1 and 2 concatenated together.

Length

Type	any
Description	vector
Compulsory	Yes
Default	

Return type: real

Returns the number of elements in the argument. The result will be 1 for a scalar and 0 for an empty value.

The Length function is the only function which will not return an error if supplied with an 'empty' value. Empty variables are returned by some functions when they cannot produce a return value. All other functions and operators will yield an error if presented with an empty value and abort any script that called it.

ListDirectory

Type	string	string
Description	Path specification	Option
Compulsory	Yes	No
Default		'none'

Return type: string array

Lists all files that comply with the spec provided in argument 1.

Argument 1

Specification for output. This would usually contain a DOS style wild card value. E.g. 'C:\Program Files\SIMetrix 42*.*'. No output will result if just a directory name is given.

Argument 2

If omitted, the result will be file names only. If set to 'fullpath', the full path of the files will be returned.

In

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Returns the natural logarithm of the argument. If the argument is real and 0 or negative an error will result. If the argument is complex it will return a complex result even if the imaginary part is 0 and the real part negative. An error will always occur if both real and imaginary parts are zero.

Locate

Type	real	real
Description	vector	search value
Compulsory	Yes	Yes
Default		

Return type: real

Function performs a binary search on the input vector (argument 1) for the value specified in argument 2. The input vector *must be monotonic* i.e. either always increasing or always reducing. This is always the case for the reference vector (see “[Vector References](#)” on [page 42](#)) of a simulation result. If the input vector is increasing (positive slope) the return value is the index of the value immediately *below* the search value. If the input vector is decreasing (negative slope) the return value is the index of the value immediately *above* the search value.

log

Identical to [log10](#) ([page 230](#)). We recommend always using `log10`. `log()` variably means `ln` or `log10` depending on the program, language etc. and it is rarely clear exactly which is meant.

log10

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Returns log to base 10 of argument. If the argument is real and 0 or negative an error will result. If the argument is complex it will return a complex result even if the imaginary part is 0 and the real part negative. An error will always occur if both real and imaginary parts are zero.

mag

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real array

Returns the magnitude of the argument. This function is identical to the abs function ([page 73](#)).

magnitude

Identical to mag - see above

MakeDir

Type	string
Description	Directory name
Compulsory	Yes
Default	

Return type: real

Creates the directory specified by arg 1. Returns 0 if successful otherwise returns 1.

MakeLogicalPath

Type	string
Description	Path
Compulsory	Yes
Default	

Return type: string

Converts a file system path to a symbolic path using the automatic path matching mechanism. This process is described in the *User's Manual* in the “Sundry Topics” chapter under the “Symbolic Path Names” section.

MakeString

Type	real	string array
Description	Number of elements in result	Initial values
Compulsory	Yes	No
Default		

Return type: string array

Creates an array of strings. Length of array is given as argument to function. The strings may be initialised by supplying argument 2.

Argument 1

Number of elements to create in string array.

Argument 2

Initialises values of string. Can be used to extend an existing string. e.g:

```
Let str = ['john', 'fred', 'bill']
Let str = MakeString(6, str)
```

In the above the string `str` will be extended from length 3 to length 6 by the call to `MakeString`.

Return Value

Returns new string

ManageMeasureDialog

Type	string
Description	
Compulsory	Yes
Default	

Return type: string array

Opens dialog box used to manage graph measurements.

Argument 1

String array defining measurements to be entered into the dialog box. Each string is a semi-colon delimited line with each element defined in the following table:

5

Token index	Description
0	Label listed in list box
1	Expression
2	Format template
3	Label as displayed on graph
4	Full description
5	Needs cursors on: 0 or 1
6	Is custom measurement: 0 or 1

Max

Type	real	real
Description	vector 1	vector 2
Compulsory	Yes	Yes
Default		

Return type: real array

Returns an array equal to the length of each argument. Each element in the array holds the larger of the corresponding elements of argument 1 and arguments 2

Maxidx

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real

Returns index of the array element in argument 1 with the largest magnitude.

Maxima

Type	real array	real array	string
Description	vector	[min limit, max limit]	options
Compulsory	Yes	No	No
Default		$[-\infty, +\infty]$	<<empty>>

Return type: real array

Returns array of values holding every maximum point in the supplied vector whose value complies with limits specified in argument 2.

Argument 1

Input vector

Argument 2

Real array of max length 2. Specifies limits within which the input values must lie to be included in the result.

0	Minimum limit i.e. maxima must be above this to be accepted
1	Maximum limit i.e. maxima must be below this to be accepted.

Argument 3

String array of max length 2. Specifies two possible options:

'xsort'	If specified the output is sorted in order of their x-values (reference). Otherwise the values are sorted in descending order of y magnitude.
'nointerp'	If <i>not</i> specified the values returned are obtained by fitting a parabola to the maximum and each point either side then calculating the x, y location of the point with zero slope. Otherwise no interpolation is carried out and the literal maximum values are returned.
'noendpts'	If specified, the first and last points in the data will not be returned as maximum points.

Return value

The function returns the XY values for each maximum point. The X-values are returned as the vector's reference (see [“Vector References”](#) on page 42).

Maximum

Type	real/complex array	real	real
Description	Vector	Min range	Max range
Compulsory	Yes	No	No
Default		start of vector	end of vector

Return type: Real

Returns the largest value found in the vector specified in argument 1 in the range of x values specified by arguments 2 and 3

mean

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Returns the average of all values in supplied argument. If the argument is complex the result will also be complex.

Mean1

Type	real array	real	real
Description	Input vector	start x value	end x value
Compulsory	Yes	No	No
Default		Start of input vector	End of input vector

Return type: real

Returns the integral of the supplied vector between the ranges specified by arguments 2 and 3 divided by the span (= arg 3 -arg 2). If the values supplied for argument 2 and/or 3 do not lie on sample points, second order interpolation will be used to estimate y values at those points.

MeasureDialog

Type	string array	string array	string array
Description	Dialog data	Initial settings	condition
Compulsory	No	No	No
Default			

Return type: string array

Opens dialog for specifying graph measurements

Argument 1

Dialog data. Format the same as for argument 1 in function [“ManageMeasureDialog” on page 232](#) except the final token is not required.

Argument 2

String array containing initial values. List in same format as return value

Argument 3

If set 'haveCursors' indicates to dialog box that graph cursors are enabled.

Return value

String array of length 10 providing user selections. Fields defined as follows:

Index	Description
0	Measurement selection from list box
1	'1' if Cursor span box is checked
2	'1' if AC coupled box is checked
3	'1' if Integral cycles box is checked
4	Graph label custom definition
5	Expression custom definition
6	'1' if Save to pre-defined box is checked
7	Format template custom definition
8	Label for custom definition
9	Long description for custom definition

MessageBox

Type	string array	string array
Description	Message	Options
Compulsory	Yes	No
Default		

Return type: string

Opens a message dialog box with a choice of styles.

Argument 1

1 or 2 element string array. First element is the text of the message to be displayed in the box. The second element is the box title. If the second element is not supplied the box title will be the name of the application - e.g. 'SIMetrix Micron AD'

Argument 2

1 or 2 element string array. First element is box style. This may be one of the following:

'AbortRetryIgnore'	Three buttons supplied for user response - Abort, Retry and Ignore
'Ok'	Ok button only
'OkCancel'	Ok and Cancel button
'YesNo'	Yes and No buttons
'YesNoCancel'	Yes, No and Cancel buttons.

Default = 'OkCancel'

Second element is icon style. A small icon is displayed in the box to indicate the nature of the message. Possible values:

'Warn'
 'Info'
 'Question'
 'Stop'

Default = 'Info'

Return value

is a single string indicating the user's response. One of:

'Abort'
 'Cancel'
 'Ignore'
 'No'
 'Ok'
 'Retry'
 'Yes'

Mid

Type	string	real	real
Description	String	Start index	Length of result
Compulsory	Yes	Yes	No
Default			to end of string

Return type: string

Returns a string constructed from a sub string of argument 1. First character is at index specified by argument 2 while argument 3 is the length of the result. The first character is at index 0.

Example

```
Mid('Hello World!', 6, 5)
```

will return 'World'.

See also

Char [page 82](#)

Minidx

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real

Returns index of the array element in argument 1 with the smallest magnitude.

Min

Type	real	real
Description	vector 1	vector 2
Compulsory	Yes	Yes
Default		

Return type: real array

Returns an array equal to the length of each argument. Each element in the array holds the smaller of the corresponding elements of argument 1 and arguments 2

Minima

Type	real array	real array	string
Description	vector	[max limit, min limit]	options
Compulsory	Yes	No	No
Default		[+∞, -∞]	<<empty>>

Return type: real array

Returns array of values holding every minimum point in the supplied vector whose value complies with limits specified in argument 2.

Argument 1

Input vector

Argument 2

Real array of max length 2. Specifies limits within which the input values must lie to be included in the result.

0	Maximum limit i.e. minima must be below this to be accepted
1	Minimum limit i.e. minima must be above this to be accepted.

Argument 3

String array of max length 2. Specifies two possible options:

'xsort'	If specified the output is sorted in order of their x-values (reference). Otherwise the values are sorted in descending order of y magnitude.
'nointerp'	If <i>not</i> specified the values returned are obtained by fitting a parabola to the minimum and each point either side then calculating the x, y location of the point with zero slope. Otherwise no interpolation is carried out and the literal minimum values are returned.
'noendpts'	If specified, the first and last points in the data will not be returned as minimum points.

Return value

The function returns the XY values for each minimum point. The X-values are returned as the vector's reference (see [“Vector References” on page 42](#)).

Minimum

Type	real/complex array	real	real
Description	Vector	Min range	Max range
Compulsory	Yes	No	No
Default		start of vector	end of vector

Return type: Real

Returns the smallest value found in the vector specified in argument 1 in the range of x values specified by arguments 2 and 3

ModelLibsChanged

No arguments

Return type: real

Returns 1 if the installed model libraries have been changed since the last call to this function. The function always returns 1 the first time it is called after program start.

Navigate

Type	string	string
Description	full component reference	path of hierarchical root
Compulsory	Yes	Yes
Default		

Return type: string

Returns path name of schematic hierarchical block.

Argument 1

Component reference of block. This must be the full reference specifying the full path to the root. For example the reference U3.U4 refers to a block of reference U4 found in the underlying schematic of a block of reference U3 in the root schematic.

Argument 2

File system pathname of root schematic.

NearestInst

Type	string
Description	property name
Compulsory	Yes
Default	

Return type: string

Returns value of property given as argument 1 for nearest instance to cursor. If the nearest instance to the cursor does not possess the specified property, an empty string will be returned.

See also

Branch [page 80](#)
NetName [page 241](#)
PinName [page 255](#)

NetName

Type	string
Description	option
Compulsory	No
Default	<<empty>>

Return type: string

Returns the net name of the nearest wire or instance pin.

The argument determines the behaviour of the function for child schematics in a hierarchy. If the argument is omitted or empty, the full net name is returned including the parents name(s). (E.g. U2.U6.R3_P). If the argument is the string 'flat' the value returned is just the local netname (E.g. R3_P).

This function is used for voltage cross-probing. The node vectors produced by the simulator always have the same name as the net so the string returned by this function is the name of the variable holding the voltage at that node.

NetNames

Type	real
Description	Schematic handle
Compulsory	No
Default	-1

No arguments

Return type: string array

Argument 1

Schematic handle as returned by the OpenSchematic function ([page 249](#)). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

Return Value

Returns an array of strings holding *all* the net names in the currently selected schematic. Returns an empty value ([page 28](#)) if the schematic is empty or can't be found.

NetWires

Type	string	real
Description	Net name	Schematic handle
Compulsory	Yes	No
Default		-1

Return type: string array

Returns wire handles of names net.

Note that this function requires that the schematic has been netlisted. This can be forced using “**Netlist** /nooutput /nodescend” if required. Note also that, for a child schematic in a hierarchy, a local netname is expected, that is without the path prefix (e.g. ‘voutn’ not ‘u1.voutn’)

Argument 1

Name of net whose wire handles are required

Argument 2

Schematic handle as returned by the **OpenSchematic** function (page 249). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

Return Value

Returns an array of strings holding the handles for all wires on the specified net. Returns an empty string if there are no wires on the net or if the net does not exist.

NewPassiveDialog

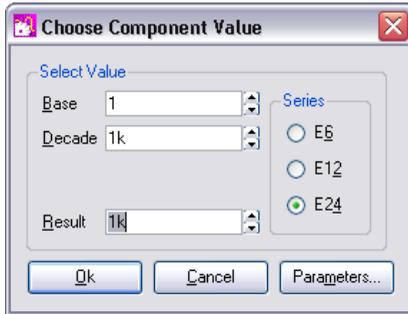
Type	string	string array	string array	string array
Description	initial value	[message, series]	parameter names	parameter values
Compulsory	Yes	No	No	No
Default		['Select value', 'E12']	<<empty>>	<<empty>>

Return type: string array

Opens a dialog box intended to select values for passive components such as resistors and capacitors. The dialog below is displayed after executing the following:

```
Let paramNames = ['temp', 'tc1', 'tc2']
```

```
Let paramValues = ['', '', '']
Show NewPassiveDialog('1k', ['Select Value', 'e24'], paramNames,
paramValues)
```



Argument 1

Initial value displayed in “Result” box. “Base” and “Decade” will be adjusted accordingly.

Argument 2

Two element string array:

0	Message displayed at the top of the box.
1	Initial setting of preferred value series. Possible values: 'E6', 'E12', 'E24'

Argument 3

String array defining list of parameter names. See argument 4.

Argument 4

String array defining list of parameter values. If arguments 3 and 4 are supplied the “Parameters...” button will be visible. This button opens another dialog box that provides the facility to edit these parameters' values.

Return value

The function returns a string array in the following form

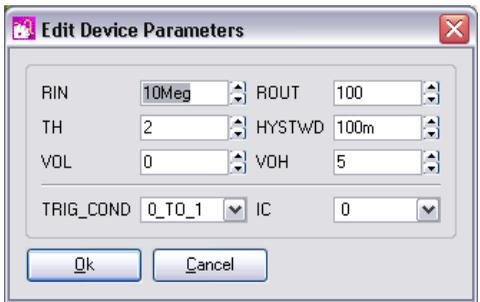
0	Value in “Result” box
1	Number of parameter values.
2 onwards	The values of the parameters in the order they were passed.

NewValueDialog

Type	string	string	string
Description	Control definitions	Initial values	Options
Compulsory	Yes	Yes	Yes
Default			

Return type: string array

General purpose user input function. A call to NewValueDialog opens a dialog box with an arbitrary number of controls of 5 different types. Any mix of the different types may be used. The following is an example with 8 controls of two different types:



Argument 1

This is a string array of length equal to the total number of controls required. Each element of the array defines the control's label, type and valid range of values. The array elements are of the form:

label[:type[:range]]

Where:

label

A text string defining the control's label. It may not contain the characters ':' or '|'.

<i>type</i>	One of:
REAL	Default if <i>type</i> omitted. Displays an edit control with an up-down spinner. Spinner increments in 1:2:5 steps.
INT or INTEGER	Displays an edit control with an up-down spinner. Spinner increments linearly with step size of 1.
STRING	Displays an edit control
BOOL	Displays a check box
LIST	Displays a drop down list with entries defined by <i>range</i> .

range Valid range of values for control delimited by '|'. Ignored for STRING and BOOL types and compulsory for LIST type. For REAL and INTEGER types, one or two values may be supplied representing the minimum and maximum valid values. The user will not be able to enter values outside this defined range. For LIST types the *range* defines the entries in the list.

Argument 2

This is a string array which must have the same number of elements as argument 1. Each element defines the initial value for the control. For BOOL types use the values “true” and “false”.

Argument 3

Function options. Currently there is only one and that is the dialog box caption.

Examples

The following call would display the dialog as shown above. Note that although this is shown here occupying several lines, the actual call must be on one line.

```
NewValueDialog(['RIN::0', 'ROUT::0', 'TH', 'HYSTWD::0', 'VOL',
'VOH', 'TRIG_COND:LIST:0_TO_1|1_TO_0', 'IC:LIST:0|1'],
['10Meg', '100', '2', '0.1', '0', '5', '0_TO_1', '0'], ['Edit
Device Parameters'])
```

norm

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Returns the input vector scaled such that the magnitude of its largest value is unity. If the argument is complex then so will be the return value.

NumDivisions

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real

Returns the number of divisions in a vector. Vectors created by multi-step runs such as Monte Carlo are sub-divided into divisions with one division per step. For a full explanation of this concept, see [“Multi-division Vectors” on page 38](#).

NumElems

Type	any
Description	vector
Compulsory	Yes
Default	

Return type: real array

Returns the number of elements in a vector. It is similar to the Length function but differs in the way it handles multi-division vectors. NumElems will return an array element for each division in the vector whereas Length will return the number of elements of the first division only.

OpenEchoFile

Type	string	string
Description	File name	Access mode
Compulsory	Yes	Yes
Default		

Return type: real

Redirects the output of the Echo ([page 377](#)) command to a file. Redirection is disabled when the CloseEchoFile function ([page 83](#)) is called or when control returns to the command line.

Argument 1

File name.

Argument 2

A single letter to determine how the file is opened. Can be either 'w' or 'a'. If 'w', a new file will be created. If a file of that name already exists, it will be overwritten. If 'a' and the file already exists, it will be appended.

OpenFile

Type	string	string
Description	File path	File open mode
Compulsory	Yes	Yes
Default		

Return type: real

Opens a file and returns its handle. This may be used by the command Echo ([page 377](#)). Use CloseFile ([page 83](#)) to close the file.

Argument 1

Path of file to open.

Argument 2

Open mode. May be 'w' or 'wa'. 'w' opens file for writing and clears the file if it already exists. 'wa' opens the file for append, that is it will append any output to the file if that file already exists.

OpenSchem

Type	string	string array
Description	File path	Options
Compulsory	Yes	No
Default		

Return type: string

Opens a schematic similar to the command [OpenSchem \(page 396\)](#) but returns a code indicating success or otherwise.

Argument 1

Schematic file path

Argument 2

Options. String array may contain any of the following:

Option	Description
'cd'	Change current working directory to the location of the specified schematic file
'readonly'	Open in read only mode
'selectiveReadOnly'	Open in read only mode if the schematic file cannot be opened for writing

Return value

The return value may be one of the following:

Code	Description
NOERR	Schematic opened successfully
SC_READONLY	Schematic file is read only. If 'readonly' or 'selectiveReadOnly' was specified as an option, then the schematic would have been successfully opened but it will not be possible to save it to the same file.
SC_LOCKED	Schematic file is in use by another SIMetrix user. If 'readonly' or 'selectiveReadOnly' was specified as an option, then the schematic would have been successfully opened but it will not be possible to save it to the same file.
FILE_NONAME	No file name was given. (Arg1 an empty string)
FILE_CANTOPENFORREAD	Can't open specified file because it doesn't exist or the path is bad

OpenSchematic

Type	string
Description	File path
Compulsory	Yes
Default	

Return type: real

Opens a schematic given its file system path. The return value may be used with a number of other functions.

Argument 1

File system path to schematic file. The schematic does not need to be currently open.

Return Value

Returns a numeric value (always an integer) that can be used for a wide range of functions that return information about a schematic. If the schematic cannot be opened for any reason, the function returns -1.

Notes

The OpenSchematic function along with the functions listed below that support schematic handles, allow information to be retrieved from schematics that are not currently open. If the specified schematic *is* open then the values returned by the supported functions will reflect the state of the *open* schematic and not the saved schematic.

The return value from OpenSchematic can be used with the following functions:

- [GetChildModulePorts \(page 147\)](#)
- [GetF11Lines \(page 160\)](#)
- [GetInstancePinLocs \(page 170\)](#)
- [HasProperty \(page 211\)](#)
- [Instances \(page 217\)](#)
- [InstNets2 \(page 219\)](#)
- [InstPins \(page 219\)](#)
- [InstPoints \(page 220\)](#)
- [InstProps \(page 221\)](#)
- [NetNames \(page 241\)](#)
- [NetWires \(page 242\)](#)
- [PropFlags \(page 257\)](#)
- [PropFlags2 \(page 258\)](#)
- [PropValues \(page 260\)](#)
- [PropValues2 \(page 261\)](#)
- [SymbolName \(page 303\)](#)
- [WirePoints \(page 319\)](#)
- [Wires \(page 320\)](#)

The handle returned by OpenSchematic may be closed using the function CloseSchematic. After a call to CloseSchematic, the handle will no longer be valid and any function it is supplied to will fail. However, it is not usually necessary to call CloseSchematic as handles are automatically closed when control returns to the command line.

Parse

Type	string	string	string
Description	Input string	Delimiters	options
Compulsory	Yes	No	No
Default		Space, tab, comma	<<empty>>

Return type: string array

Splits up the string supplied as argument 1 into substrings or tokens. The characters specified in argument 2 are treated as separators of the substrings. For example, the following call to Parse():

```
Parse('c:\simetrix\work\amp.sch', '\')
```

returns:

```
'c:'  
'simetrix'  
'work'  
'amp.sch'
```

If the second argument is omitted, spaces and tab characters will be treated as delimiters. If a space is include in the string of delimiters, tab characters will be automatically added.

If the third arguments is present and equal to 'quoted' the function will treat strings enclosed in double quotes as single indivisible tokens.

ParseAnalysis

Type	string
Description	Analysis spec
Compulsory	Yes
Default	

Return type: string array

Opens the choose analysis dialog initialised according to the analysis controls passed as the argument. Returns a new analysis spec that may be passed to a netlist

Argument 1

Analysis spec as it would appear in a netlist or the F11 window. E.g. lines beginning with .TRAN, .AC, .DC etc.

Return Value

String array of length 2. Element 0 contains the new analysis spec. Note individual simulator controls are separated by new line characters.

Element 1 identifies how the user closed the dialog box as defined below:

```
Run button      '2'
Cancel button   '1'
OK button       '0'
```

ParseParameterString

Type	string	string array	string	string	string
Description	String to parse	Parameter names to process	action	Write value	Options
Compulsory	Yes	Yes	Yes	No	No
Default					

Return type: string array or scalar

Parses a string of name-value pairs and performs some specified action on them. The function can read specified values and return just the values. It can write to specific values and return a modified string. It can also delete specific values.

Argument 1

String to parse. This is a list of name-value pairs but may also contain any number of unlabelled values at the start of the string. The number of unlabelled values must be specified in argument 3 (see below). Examples:

Without any unlabelled value:

```
W=1u L=2u AD=3e-12 AS=3e-12
```

With 1 unlabelled value

```
2.0 DTEMP=25.0
```

The above shows an equals sign separating names and values, but these may be omitted.

Argument 2

String array listing the names to be processed. If reading (see below) only the values of the names supplied here will be returned. If writing, the names listed in this argument will be edited with new values supplied in argument 4. If deleting, these names will be removed.

Unlabelled parameters may be referenced using the special name '\$unlabelled\$' followed by the position. I.e. the first unlabelled parameter is position 1, the second 2 and so on. So '\$unlabelled\$1' refers to the first unlabelled parameter.

Argument 3

1 or 2 element string array. The first element is the action to be performed. The second element is the number of unlabelled parameters that are expected in the input string. This is zero if omitted.

Argument 4

Values to write. These have a 1:1 correspondence with the parameter names in argument 2.

Argument 5

If set to 'allowquoted', the function will treat any items enclosed in single or double quotation marks as a single token even if there are spaces within.

Return Value

If reading, the return value is an array of strings holding the values of the specified parameters. Otherwise it the input string appropriately modified according to the defined action.

Examples

This will return the string array ['1u', '2u']:

```
Let str = 'W=1u L=2u AD=3e-12 AS=3e-12'  
ParseParameterString(str, ['W', 'L'], 'read')
```

This returns '2.0'

```
Let str = '2.0 DTEMP=25.0'  
ParseParameterString(str, '$unlabelled$1', ['read', '1'])
```

This will return the modified string: 'W=90n L=120n AD=3e-12 AS=3e-12'

```
Let str = 'W=1u L=2u AD=3e-12 AS=3e-12'  
ParseParameterString(str, ['W', 'L'], 'write', ['90n', '120n'])
```

This will return the modified string: 'AD=3e-12 AS=3e-12'

```
Let str = 'W=1u L=2u AD=3e-12 AS=3e-12'  
ParseParameterString(str, ['W', 'L'], 'delete')
```

ParseSimplisInit

Type	string
Description	Simplis init file
Compulsory	Yes
Default	

Return type: string array

Reads and parses the .init file created by a SIMPLIS run. This is used by the feature that back-annotates SIMPLIS schematics with initial condition values.

PathEqual

Type	string array	string array
Description	Path 1	Path 2
Compulsory	Yes	Yes
Default		

Return type: real array

Compares two string arrays and returns a real array of the same length with each element holding the result of a string comparison between corresponding input elements. The string comparison assumes that the input arguments are file system path names and will choose case sensitivity according to the underlying operating system. On Linux the comparison will be case sensitive and on Windows it will be case insensitive.

Argument 1

First pathname or pathnames to be compared.

Argument 2

Second pathname or pathnames to be compared.

Return Value

Real array of the same length as the arguments. If the lengths of the arguments are different, an empty vector will be returned. Each element in the array will be either -1, 0, or +1. 0 means the two strings are identical (subject to case sensitivity as described above).

ph

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real array

Returns the phase of the argument in degrees.

Each of the function `ph()`, `phase()` and `phase_rad()` produce a continuous output i.e. it does not wrap from 180 degrees to -180 degrees.

This function always returns a result in degrees. This has changed from versions 3.1 and earlier which returned in degrees or radians depending on the setting of the 'Degrees' option. For phase in radians, use `phase_rad()`.

phase

Identical to `ph` - see above.

phase_rad

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real array

Identical to `ph` and `phase` functions (see above) except that the result is in radians.

PhysType

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: string

Returns the physical type of the argument. Possible values are.

" (meaning dimensionless quantity)
 'unknown'
 'Voltage'
 'Current'
 'Time'
 'Frequency'
 'Resistance'
 'Conductance'
 'Capacitance'
 'Inductance'
 'Energy'
 'Power'
 'Charge'
 'Flux'
 'Volt^2'
 'Volt^2/Hz'
 'Volt/rtHz'
 'Amp^2'
 'Amp^2/Hz'
 'Amp/rtHz'
 'Volts/sec'

See also

Units [page 313](#)

PinName

Type	string	string
Description	options	property name
Compulsory	No	No
Default	<<empty>>	<<empty>>

Return type: string array

Returns information about the schematic instance pin nearest the mouse cursor. The format of the result depends on the values of the arguments.

Argument 1

May be one of five possible values:

<<empty>>	Return value is full hierarchical name of pin. (e.g. U1.U6.Q1#c)
'flat'	Return value is local name without hierarchical prefix. (e.g. Q1#c)
'property'	Return value is string array with a pair of elements for each pin at the location. First value in each pair is the value of the property specified in argument 2 and the second is the pin number.
'distance'	Return value has two elements. The second element is the distance of the cursor to the pin in "sheet units". There are 120 "sheet units" per grid at X 1 magnification.
['flat', 'distance']	As 'distance' but returns local net name without hierarchical prefix.

Argument 2

Property name whose value is returned if argument 1 is 'property'. See above.

Progress

Type	real	string array
Description	Position of progress bar in %	options/control
Compulsory	Yes	No
Default		<<empty>>

Return type: real

Opens a dialog box showing a progress bar.

Argument 1

Value from 0 to 100 specifying the position of the bar.

Argument 2

String array of max length 2 used to specify options and control as follows:

'open'	Box is displayed (cannot be used with 'close')
'close'	Box is hidden (cannot be used with 'open')
'showabort'	If specified an abort button will be displayed

Return value

The function returns a two element array. The first element returns the value of argument 1, while the second returns 1 if the abort button has been pressed. If the abort button has not been pressed, the second element returns 0.

Probe

No arguments

Return type: real

Changes schematic cursor to a shape depicting an oscilloscope probe. Returns when the user presses a mouse key. If the left key is pressed return value is 1 otherwise it is 0. Probe returns on both up and down strokes of mouse key. See “[Example 3: Cross probing](#)” on [page 22](#) for an example of using the Probe function.

ProcessingAccelerator

No arguments

Return type: real

Returns 1.0 if the script being executed was called by activating a menu accelerator. Otherwise returns 0.0. This functions makes it possible to modify the action taken if the user activates a function via an accelerator key rather than the menu.

PropFlags

Type	string	string	string	real
Description	property name (for flags)	property name (for id)	property value (for id)	Schematic handle
Compulsory	Yes	No	No	No
Default		Selected components	All instances with property name in argument 2	-1

Return type: string array

Returns the attribute flags for instances identified by arguments 2 and 3. See “[Attribute flags](#)” on [page 404](#) for details.

This function has been superseded by PropFlags2 (see below) and it is not recommended for new scripts. PropFlags2 has rearranged arguments allowing the schematic handle to be specified without requiring the property value to provided. It also has more convenient behaviour in the situation when there is no instance match.

Argument 1

Property whose flags are to be returned.

Argument 2/3

If present these arguments identify the instances to be examined. If only argument 2 is specified then all instances on the current schematic that possess that property will be used. If argument 3 is also present then the instance name and value must match argument 2 and 3 respectively. If neither are present the selected instances will be used.

Argument 4

Schematic handle as returned by the OpenSchematic function ([page 249](#)). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

Return value

The function returns a string array of length equal to the number of instances identified by arguments 2 and 3. Each element will hold a flag value for the property specified in argument 1.

The function will return an empty vector ([page 28](#)) if the specified schematic could not be found. If no instance matches arguments 2 and 3, an empty *string* will be returned.

PropFlags2

Type	string	real	string	string
Description	property name (for flags)	Schematic handle	property name (for id)	property value (for id)
Compulsory	Yes	No	No	No
Default		-1	Selected components	All instances with property name in argument 2

Return type: string array

Returns the attribute flags for instances identified by arguments 3 and 4. See “[Attribute flags](#)” on [page 404](#) for details.

This function replaces PropFlags. Its behaviour is similar but the arguments have been rearranged and its behaviour in the event of no instance match is different.

Argument 1

Property whose flags are to be returned.

Argument 2

Schematic handle as returned by the `OpenSchematic` function ([page 249](#)). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

Argument 3/4

If present these arguments identify the instances to be examined. If only argument 3 is specified then all instances on the current schematic that possess that property will be used. If argument 4 is also present then the instance name and value must match argument 3 and 4 respectively. If neither are present the selected instances will be used.

Return value

The function returns a string array of length equal to the number of instances identified by arguments 3 and 4. Each element will hold a flag value for the property specified in argument 1.

Note that this function compliments `PropValues2` ([page 261](#)) and `SymbolNames` ([page 304](#)) and will return the same number of values and in the same order, provided the same instance identifying arguments are given.

The function will return an empty *vector* ([page 28](#)) if no instances match arguments 3 and 4. This differs from `PropFlags` which returns an empty *string* in this situation. The behaviour of `PropValues2` is much more convenient and it is recommended that this is used in all new scripts.

`PropFlags2` will also return an empty vector if the specified schematic could not be found.

PropValue

Type	string
Description	Property name
Compulsory	Yes
Default	

Return type: string

Returns the value of the property supplied as an argument for the selected component. If no components are selected or more than one component is selected, an empty string will be returned.

PropValues

Type	string	string	string	real
Description	Property name whose value is required	Property name to identify instance	Property value to identify instance	Schematic handle
Compulsory	Yes	No	No	No
Default		Use selected components if omitted	All instances with property name in arg2	-1

Return type: string array

Returns a property value for instances identified by arguments 2 and 3.

This function has been superseded by PropValues2 (see below) and it is not recommended for new scripts. PropValues2 has rearranged arguments allowing the schematic handle to be specified without requiring the property value to be provided. It also has more convenient behaviour in the situation when there is no instance match.

Argument 1

Property whose value is to be returned.

Argument 2/3

If present these arguments identify the instances to be examined. If only argument 2 is specified then all instances on the specified schematic that possess that property will be used. If argument 3 is also present then the instance name and value must match argument 2 and 3 respectively. If neither are present the selected instances will be used.

Argument 4

Schematic handle as returned by the OpenSchematic function ([page 249](#)). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

Return value

The function returns a string array of length equal to the number of instances identified by arguments 2 and 3. Each element will hold a value for the property specified in argument 1.

The function will return an empty vector ([page 28](#)) if the specified schematic could not be found. If no instance matches arguments 2 and 3, an empty *string* will be returned.

PropValues2

Type	string	real array	string	string
Description	Property name whose value is required	Schematic handle and sort option	Property name to identify instance	Property value to identify instance
Compulsory	Yes	No	No	No
Default		-1	Use selected components if omitted	All instances with property name in arg2

Return type: string array

Returns a property value for instances identified by arguments 3 and 4.

This function replaces PropValues. Its behaviour is similar but the arguments have been rearranged and its behaviour in the event of no instance match is different and more convenient.

Argument 1

Property whose value is to be returned.

Argument 2

First element is a schematic handle as returned by the OpenSchematic function ([page 249](#)). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

A second element may be supplied and if non-zero, the results will be sorted by location. Otherwise they will not be sorted.

Argument 3/4

If present these arguments identify the instances to be examined. If only argument 2 is specified then all instances on the specified schematic that possess that property will be used. If argument 3 is also present then the instance name and value must match argument 2 and 3 respectively. If neither are present the selected instances will be used.

Return value

The function returns a string array of length equal to the number of instances identified by arguments 2 and 3. Each element will hold a value for the property specified in argument 1.

Note that this function is analogous to the functions PropFlags2 ([page 258](#)) and SymbolNames ([page 304](#)) and for identical values of arguments 3 and 4 will return an array of the same length and in the same order.

The function will return an empty *vector* if no instances match arguments 3 and 4. This differs from PropValues which returns an empty *string* in this situation. The behaviour of PropValues2 is much more convenient and it is recommended that this is used in all new scripts.

PropValues2 will also return an empty vector if the specified schematic could not be found.

PutEnvVar

Type	string
Description	Definition
Compulsory	Yes
Default	

Return type: real

Write a system environment variable. Note that this only modifies environment variables in the current process and any child processes initiated using the Shell or ShellOld commands.

Argument 1

Definition. Must be of form *name=value*

Return value

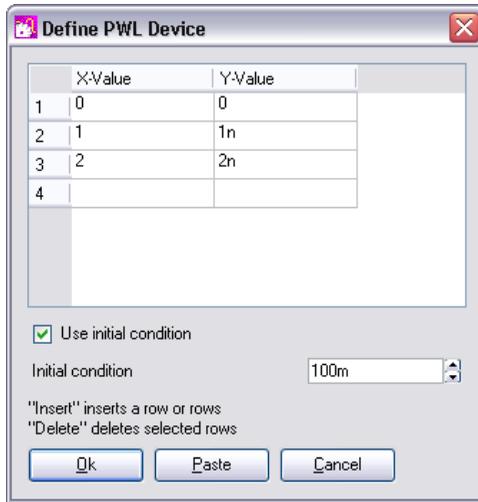
The function returns 1 on success or 0 on failure. Failure can occur if the argument is of the wrong format.

PWLDialog

Type	string	string
Description	X-Y Pairs	Options
Compulsory	Yes	No
Default		

Return type: string array

Opens the dialog box shown below allowing the entry of X-Y pairs intended for the definition of piece-wise linear devices.

**Argument 1**

X-Y Pairs to initialise box. The above example would be displayed after a call to:

```
Show pwldialog([0,0.5,1,1.5,2,2.5])
```

Argument 2

Up to three element string array to define box labels:

Index	Description
0	Box caption. Default: 'Define PWL Source'
1	Label for X-Values column. Default: 'Time'
2	Label for Y-Values column. Default: 'Value'
3	Initial condition mode. Maybe: <ul style="list-style-type: none"> 'none': Default setting. No initial condition displayed 'segment': Initial segment. Initial condition value is an integer with a minimum value of 1 and a maximum value equal to the number of rows. (Used for some SIMPLIS PWL devices). Use initial condition check box will not be shown 'continuous': Initial condition is a non-integral number. Use initial condition check box will be shown

Argument 3

Real array with two elements. First element is the initial state of the 'Use initial condition' check box. Second element is the initial value of the initial condition edit box.

This argument is ignored if initial condition mode is set to 'none'

Return Value

The function returns the X-Y Pairs entered by the user in the same format as for argument 1. If initial conditions were enabled on input, there will be two additional elements at the end. The first will be either 'true' or 'false' to indicate whether 'Use initial condition' was checked and the second is the value of the initial condition.

QueryData

Type	string array	string array
Description	Data	Filter
Compulsory	Yes	Yes
Default		

Return type: string array

Filters a list of data items according to search criteria.

Argument 1

The data to be filtered. This should consist of an array of strings comprising semi-colon delimited fields. The filter supplied in argument 2 matches each field to certain criteria and returns the data in the output if those criteria are satisfied.

Argument 2

Filter to determine if data in arg 1 is passed to the output. The filter consists of one or more semi-colon delimited lists which can be combined in Boolean combinations. Each of the lists is compared with the input data for a match and if the resulting Boolean expression is true, the data item is accepted and passed to the return value. Wild cards '*' and '?' may be used in any field. The system is best explained with examples.

Suppose a data item in arg 1 is as follows.

```
IRFI520N;nmos_sub;X;NMOS;;;SIMetrix
```

and the filter supplied in arg 2 is:

```
*;*;X;*;*;*;*;SIMetrix
```

This will match successfully. The third and last fields are the same in both the data and the filter and the remaining filter fields are the '*' wild card which means that anything will be accepted in the corresponding data field. With the following filter, however, the data will not be accepted:

```
*;*;X;*;*;*;SIMPLIS
```

Here the last field doesn't match.

In the above simple examples, only one filter list has been supplied. However, it is possible to use more sophisticated filters consisting of multiple lists combined using Boolean operators. Boolean operators are specified with the key words:

```
\OR
\AND
\XOR
\NOT
```

These can be used to make a Boolean expression using “reverse polish” notation. Here is an example:

```
[ '*;nmos;*;*;*;*;SIMetrix',
  '*;nmos_sub;*;*;*;*;SIMetrix', '\OR' ]
```

This will accept any data where the last field is 'SIMetrix' and the second field is either 'nmos' or 'nmos_sub'. Note that the keyword '\OR' is applied after the filter lists.

As well as the '*' wild card, the '?' may also be used. '?' matches only a single character whereas '*' matches any number of characters. For example:

```
?mos
```

Would match 'pmos' as well as 'nmos'. It would also match any other four letter word that ended with the three letters 'mos'.

Return Value

String array of length up to but not exceeding the length of argument 1. Contains all arg 1 items that match the filter as explained above.

RadioSelect

Type	real	string	string	string
Description	Number of button initially selected	Button labels		Help context id
Compulsory	No	No	No	No
Default	1	empty	Dialog box caption	

Return type: real

Opens a dialog box with up to 6 radio buttons. The number of buttons visible depends on the length of argument 2. All six will be displayed if it is omitted.

The labels for each button is specified by argument 2. The button initially selected is specified by argument 1. Argument 3 provides the text in the dialog boxes caption bar.

The return value identifies the selected button with the top most being 1. If the user cancels the function returns 0.

Argument 4 specifies a help context id and if present a Help button will be displayed. This is used by some internal scripts.

See also

BoolSelect [page 79](#)

EditSelect [page 79](#)

ValueDialog [page 316](#)

NewValueDialog [page 244](#)

Range

Type	real/complex or string array	real	real
Description	vector	start index	end index
Compulsory	Yes	No	No
Default		0	Vector length-1

Return type: matches argument 1

Returns a vector which is a range of the input vector in argument 1. The range extends from the indexes specified by arguments 2 and 3. If argument 3 is not supplied the range extends to the end of the input vector. If neither arguments 2 or 3 are supplied, the input vector is returned unmodified.

See also the Truncate function on [page 312](#).

re

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real array

Returns the real part of the complex argument.

ReadClipboard

No arguments

Return type: string array

Returns text contents of the windows clipboard. Data is returned as one line per array element.

Note that the Show command ([page 426](#)) can be used to write to the clipboard.

ReadConfigCollection

Type	string
Description	Section
Compulsory	Yes
Default	

Return type: string array

Returns the contents of an entire section in the configuration file. Note that only the values are returned, not the names of the keys. To get the names of the keys, use “[ReadConfigSetting](#)” (see below) with an empty second argument

Argument 1

Name of section to return.

Return Value

An array of strings holding the values for every entry in the specified section. Note that the key names are not returned. This function is intended to be used for managing lists of values identified by their section name. Use [AddConfigCollection](#) ([page 74](#)) to write values to the list.

ReadConfigSetting

Type	string	string
Description	Section	Key
Compulsory	Yes	No
Default		

Return type: string or string array

Reads a configuration setting. Configuration settings are stored in the configuration file. See ‘Configuration Settings’ in Chapter 13 of the *User’s Manual* for more information. Settings are defined by a key-value pair and are arranged into sections. The function takes the name of the key and section and returns the value.

Note that option settings (as defined by the Set command) are placed in the ‘Options’ section. Although these values can be read by this function this is not recommended and instead you should always use the [GetOption \(page 183\)](#) function.

Argument 1

Section name. See description above for explanation.

Argument 2

Key name. See description above for explanation.

If this argument is omitted, the function will return a list of all keynames found in the specified section.

Return Value

Value read from configuration file.

See Also

[WriteConfigSetting \(page 321\)](#)

ReadFile

Type	string
Description	File name
Compulsory	Yes
Default	

Return type: string array

Returns an array of strings holding lines of text from the file specified by argument 1. The file is expected to contain only ASCII text. The operation will be aborted if non-ASCII characters are encountered.

ReadIniKey

Type	string	string	string
Description	Inifile name	Section name	Key name
Compulsory	Yes	Yes	Yes
Default			

Return type: string array

Reads an INI file. An INI file usually has the extension .INI and is used for storing configuration information. INI files are used by many applications and follow a standard format as follows:

```
[section_name1]
key1=value1
key2=value2
...
[section_name2]
key1=value1
key2=value2
...
etc.
```

There may be any number of sections and any number of keys within each section.

The ReadIniKey function can return the value of a single key and it can also return the names of the all the keys in a section as well as the names of all the sections.

Argument 1

File name. You should always supply a full path for this argument. If you supply just a file name, the system will assume that the file is in the WINDOWS directory. This behaviour may be changed in future versions. For maximum compatibility, always use a full path.

Argument 2

Section name. If this argument is an empty string, the function will return the names of the sections in the file.

Argument 3

Key name. If this argument is an empty string and argument 2 is not an empty string, the function will return the names of all the keys in the named section.

ReadRegSetting

Type	string	string	string
Description	Key name	Value name	Top level tree
Compulsory	Yes	Yes	No
Default			'HKCU'

Return type: string

Reads a string setting from the windows registry. Currently this function can only read settings in the HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE top level trees.

On Linux the registry is simulated using settings in the SIMetrix configuration file.

Argument 1

Name of key. This must be a full path from the top level. E.g. 'Software\SIMetrix\'

Argument 2

Name of value to be read

Argument 3

Top level tree. This may be either 'HKEY_CURRENT_USER' or 'HKEY_LOCAL_MACHINE' or their respective abbreviations 'HKCU' and 'HKLM'.

Return Value

Returns value read from the registry. If the value doesn't exist, the function returns an empty vector.

ReadSchemProp

Type	string	string
Description	Property name	Schematic path
Compulsory	Yes	No
Default		Currently displayed

Return type: string

Returns value of schematic window property value.

Argument 1

Property name. There are three built-in properties that are always available. Others can be created with the WriteSchemProp function ([page 324](#)). The three built-in properties are:

'Path'	Read-only. File system path name of schematic
'RootPath'	Read/Write. Path of root in hierarchy. Value displayed in status bar of schematic
'Reference'	Read/Write. Full component reference of block representing schematic.
'Readonly'	Read-only. Readonly status of schematic. Return value may be 'TRUE' or 'FALSE'
'UserStatus'	Read/Write. Contents of user status box at the bottom of the schematic. This is currently the 4th box from the left.
'UserVersion'	Read-only. Current version number of schematic. This is updated each time the schematic is saved

Argument 2

Path of schematic to process. This must be a schematic that is currently displayed; the function can not operate on a closed schematic. If not specified, the currently selected schematic will be processed.

Return Value

Returns the value of the property

RemoveSymbolFiles

Type	string array
Description	Symbol files to remove
Compulsory	Yes
Default	

Return type: real

Uninstalls symbol library files

Argument 1

List of files to be removed. These may be symbolic paths as described in the 'Sundry Topics' chapter of the *User's Manual*.

Return Value

Returns number of files successfully removed.

real

Identical to re ([page 266](#)).

Ref

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Returns the reference of the argument. See [“Vector References” on page 42](#).

RefName

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: string

Returns the name of the reference of the supplied vector. See [“Vector References” on page 42](#). Note that the **Ref** function (above) returns the actual data for the reference.

RelativePath

Type	string	string
Description	Full path name	Reference directory
Compulsory	Yes	No
Default		Current directory

Return type: string

Returns a path relative to the reference directory (argument 2 or current working directory) of the full path name supplied in argument 1.

See also

FullPath [page 141](#)

SplitPath [page 297](#)

RemoveConfigCollection

Type	string	string array
Description	Section name	Items to remove
Compulsory	Yes	
Default		

Return type: real

Removes one or more entries from a configuration file collection.

Argument 1

Section where entries to be removed are located

Argument 2

List of strings to remove from the collection.

RemoveModelFile

Type	string array
Description	Model path names
Compulsory	Yes
Default	

Return type: string

Uninstalls the model library paths specified in the argument.

ResolveGraphTemplate

Type	string	string	string
Description	Graph object id	Template	Options
Compulsory	Yes	Yes	No
Default			

Return type: string

Evaluate template string used by graph object.

Argument 1

ID of graph object whose properties are to be used in the template. See [“Graph Object Identifiers - the “ID”” on page 449](#)

Argument 2

Template string. This can consist of literal text, properties enclosed with ‘%’ and expressions enclosed with ‘{’ and ‘}’. The property values are those belong to the object supplied in argument 1. Properties available for the various types of graph object are described in [“Objects and Their Properties” on page 450](#). Some properties return the id of another graph object. These can be used to create nested property definitions. For example %curve:label% when applied to a curve marker object returns the label of the attached curve.

The template string may also contain the special keywords <if>, <ifd>, <|> and <repeat>. These behave the same and have identical syntax as the keywords of the same name used for schematic TEMPLATE properties described in the *User’s Manual*.

Argument 3

Options. Currently there is only 1 and that is the action to take when an expression fails to evaluate. Possible values are:

‘msg’	Requires a second arg 3 to have two elements. Returns error message specified in second element of string.
‘empty’	Returns an empty value on error
‘literal’ (default)	Returns the literal text of the expression

Return value

Returns the result of evaluating the template.

Notes

This function along with [ResolveTemplate \(page 275\)](#) are implemented using the same internal program code that implements the schematic TEMPLATE property in a netlist generation and behaves in the same way.

ResolveTemplate

Type	string	string	string
Description			
Compulsory	Yes	Yes	Yes
Default			

Return type: string

Evaluate template string.

Argument 1

Template string. This can consist of literal text, expressions enclosed in ‘{’ and ‘}’ and special property names enclosed in ‘%’. The property names and their respective values may be defined in arguments 2 and 3. Properties names are substituted with their values by this function.

The template string may also contain the special keywords <if>, <ifd>, <tr> and <repeat>. These behave the same and have identical syntax as the keywords of the same name used for schematic TEMPLATE properties described in the *User’s Manual*.

Argument 2

Property names

Argument 3

Property values corresponding to property names given in argument 2.

Return value

Returns the result of evaluating the template.

Notes

This function along with [ResolveGraphTemplate \(page 274\)](#) are implemented using the same internal program code that implements the schematic TEMPLATE property in a netlist generation and behaves in the same way.

RestartTranDialog

Type	real
Description	Initial stop time
Compulsory	Yes
Default	

Return type: real

Opens a dialog box allowing the user to specify a new stop time for a transient analysis. The value is initialised with the argument. The return value is the stop time entered by the user. The user will not be able to enter a value less than that supplied in the argument.

Rms

Type	real array
Description	vector
Compulsory	Yes
Default	

Return type: real array

Returns a vector of the accumulative rms value of the input. Unlike RMS1 this function returns a vector which can be plotted.

RMS1

Type	real array	real	real
Description	vector	start x value	end x value
Compulsory	Yes	No	No
Default		start of input vector	end of input vector

Return type: real

Returns the root mean square value of the supplied vector between the ranges specified by arguments 2 and 3. If the values supplied for argument 2 and/or 3 do not lie on sample points, second order interpolation will be used to estimate y values at those points.

rnd

Type	real array
Description	vector
Compulsory	Yes
Default	

Return type: real array

Returns a vector with each element a random value between 0 and the absolute value of the argument's corresponding element.

RootSumOfSquares

Type	real	real	real
Description	vector	start x value	end x value
Compulsory	Yes	No	No
Default		start of input vector	end of input vector

Return type: real array

Similar to RMS1 function ([page 276](#)) but returns the root of the sum without performing an average.

SaveSpecialDialog

Type	string array
Description	initial values
Compulsory	No
Default	

Return type: string array

Opens the dialog used by the schematic's Save Special... menu.

Argument 1 and Return Value

A length three array of strings. The elements are defined as:

Index	Description
0	Filename
1	ASCII format? '1' or '0'
2	Save complete component? '1' or '0'

Scan

Type	string	string	real
Description	string to scan	delimiter	Min number of return values
Compulsory	Yes	No	No
Default			

Return type: string array

Splits a character delimited string into its components (known as tokens). Returns result as string array.

Character used as delimiter may be passed as argument 2. If argument 2 omitted delimiter defaults to a semi-colon.

Argument 1

String to scan

Argument 2

Delimiter. Semi-colon if omitted. Only a single character is permitted. To scan with multiple delimiters, see [“Parse” on page 250](#)

Argument 3

If present, forces the result to be a minimum size. For example, if the input string had two tokens but this argument was set to three, the result would be a string array of length 3 with the third element an empty string. In many applications, this can save testing the length of the return value to determine if an optional token was provided.

Return value

Returns tokens as an array of strings. Empty fields are treated as a separate token. E.g. in 'BUF04;buf;;Buffers;' the double semi-colon after 'buf' would return an empty entry in the returned array. So:

```
Scan('BUF04;buf;;Buffers;;')
```

would return:

```
[ 'BUF04', 'buf', '', 'Buffers', '' ]
```

ScriptName

No arguments

Return type: string

Returns the name of the currently executing script.

Search

Type	string array	string array	string
Description	list to search	items to search in list	Options
Compulsory	Yes	Yes	No
Default			

Return type: real array

Searches a list of strings for one or more items supplied in argument 1 for the item(s) supplied in argument 2. Function returns a real array of length equal to the length of argument 2. The return value is an array of reals.

Argument 1

List to search.

Argument 2

Items to search in list.

Argument 3

Set to 'path' if the items being searched are file system paths. This makes the search case sensitive on systems that use case sensitive path names - i.e. Linux. On Windows, which does not use case sensitive paths, this argument has no effect.

Return value

Array of indexes into argument 1 for the items found in argument 2. If a string in argument 2 is not found, the return value for that element will be -1.

SearchModels

Type	string
Description	File system tree to search
Compulsory	Yes
Default	

Return type: string array

This is a special purpose function designed for use with the model installation system. It returns an array of strings holding pathnames with wildcards of directories containing files with SPICE compatible models. The argument specifies a directory tree to search. The function will recurse through all sub directories of the supplied path.

Note that if the root directory of a large disk is specified, this function can take a considerable time to return. It can however be aborted by pressing the escape key.

Seconds

No arguments

Return type: real

Returns the number of seconds elapsed since January 1, 1970. Returned value is an integer.

SelectAnalysis

No arguments

Return type: real

This is a special purpose function. It opens the 'Choose Analysis' dialog box. The return value from this function is simply determined by how the user closes the box. The main operation of the dialog box happens independently of the function call mechanism. Return values are:

No schematic	3
Run button	2
Cancel button	1
OK button	0

The dialog box will not open if there is no current schematic.

The function reads the schematic's text window and translates any analysis controls present including any preceded by a single comment character. It uses the information gained to initialise the dialog box's controls. After the user has made a selection and closed the box, the controls in the schematic text window are updated. This mechanism means that analysis modes are stored with a schematic. Also, the user is free to select analysis modes by manually editing the controls in the text window. Any such changes will be reflected in subsequent calls to SelectAnalysis.

SelectColourDialog

Type	string
Description	Initial colour spec.
Compulsory	No
Default	Spec. for BLACK

Return type: string

Opens a dialog box allowing the user to define a colour. The box is initialised with the colour spec. supplied as an argument. The function returns the new colour specification.

If the user cancels the box, the function returns an empty vector.

Colour specifications are strings that provide information about the RGB content of the colour. Colour specifications should only be used with functions and commands that are designed to accept them. The format of the colour specification may change in future versions.

SelectColumns

Type	string array	real	string
Description	input data	field number	delimiter
Compulsory	Yes	Yes	No
Default			','

Return type: string array

Accepts an array of character delimited strings and returns an array containing only the specified field. This function was developed for the parts browser mechanism but is general purpose in nature.

Example

Data input (arg 1):

```
BUF600X1;Buf;;Buffers;;2,1,4,3
BUF600X2;Buf;;Buffers;;2,1,4,3
BUF601X1;Buf;;Buffers;;2,1,4,3
BUF601X2;Buf;;Buffers;;2,1,4,3
```

Field number (arg2)

```
0
```

Returns:

BUF600X1
BUF600X2
BUF601X1
BUF601X2

SelectCount

Type	string
Description	Type of item to count. 'Wires', 'Instances', 'All'
Compulsory	No
Default	'all'

Return type: real

Returns number of items selected. If argument is 'Wires', only selected wires will be counted, if argument is 'Instances', selected instances will be counted. Otherwise all items are counted.

SelectDevice

Type	string array	string	string array
Description	parts data	Selected device	User installed models
Compulsory	Yes	No	No
Default		No device selected	

Return type: real

Opens parts browser dialog. Argument is array of strings containing parts database. This is usually read from the file 'OUT.CAT' in the script directory. The format for this file is described in the "Device Library" Chapter of the *User's Manual*. Each line contains up to 8 semi-colon delimited fields. Only the first field (part number) and the fourth field (category) are displayed to the user but the values of any other field will be returned in the result.

If argument 2 is supplied and is the part number of a device included in arg 1, that device will be selected.

Argument 3 contains a list of model names that will appear in the '* User Models *' category. These will also appear in the '* Recently Installed Models *' category if the

model was installed within the last 30 days or other duration defined by the NewModelLifetime option setting.

Return value is a string array of length 8 containing the value of each field of the selected device or an empty vector if cancelled.

SelectDialog

Type	string array	string array
Description	options	list box entries
Compulsory	Yes	Yes
Default		

Return type: real array

Opens a dialog box containing a list box. The list box is filled with string items supplied in argument 2. The return value is the index or indexes of the items in the list box selected by the user.

This function is used by a number of the standard menus.

There are a number of options available and these are specified in argument 1. This is an array of strings of length up to 7. The meaning of each element is as follows:

Index	Possible values	Description
0		Dialog box caption
1		Message above list box
2	'Multiple', 'Single'	If 'single', only one item may be selected. Otherwise any number of items can be selected.
3	'Sorted', ''	If 'sorted', items in list are arranged in alphabetical order. Otherwise they are in same order as supplied.
4		Index of item to select at start. Only effective if 'single' selected for index 2. This is an integer but must be entered as a string e.g. '2'.
5		Initial string in edit box
6		Default return value if none selected

The function return value is empty if the user cancels.

Example

```
SelectDialog(['Caption', 'Message', 'single', '', '1'],  
            ['Fred', 'John', 'Bill'])
```

Will place strings 'Fred', 'John' and 'Bill' in the list box with 'John' selected initially. The strings will be in the order given (not sorted).

Select2Dialog

Type	String array	String array
Description	Initial values	List entries
Compulsory	No	No
Default		

Return type: String array

Opens a dialog box with two list boxes allowing the user to select two values

Argument 1

5 element string array. Values as follows:

Index	Description
0	List box 1 initial selection
1	List box 2 initial selection
2	Message at top of box
3	Message under left hand list box
4	Message under right hand list box

Argument 2

2 element array. The first element carries the items to be placed in the left hand list box. The second element carries the items to be placed in the right hand list box. Items are separated by a pipe ('|') symbol.

Return value

Two element array. First element carries the selected value from the left hand list box while the second value holds the selected value from the right hand list box.

SelectedProperties

Type	String
Description	Property name
Compulsory	No
Default	'Handle'

Return type: string array

Returns information about selected properties.

Argument 1

Property whose value will be used to identify the instance that possesses the selected property.

Return Value

Returns an array of length equal to 3 times the number of properties selected. Currently, however, it is only possible to select one property at a time so the return value will be either of length zero or length 3. The elements in each group of three are as defined in the following table:

Index	Description
0	Value of instance property identified in argument 1. This is used to identify the instance that possesses the selected property.
1	Name of selected property
2	Value of selected property

Notes

Properties can only be selected if the 'selectable' attribute is enabled.

SelectedWires

No arguments

Return type: string array

Returns an array of strings holding the handles of selected wires.

SelectFontDialog

Type	string	string
Description	Initial font spec.	Name of object being edited
Compulsory	No	No
Default	Default font	

Return type: string

Opens a dialog box allowing the user to define a font. The box is initialised with the font spec. supplied as an argument. The function returns the new font specification.

A second argument may be specified to identify the name of the object whose font is being edited. This is so that its font may be updated if the user presses the Apply button in the dialog box.

If the user cancels the box, the function returns an empty vector.

Font specs are strings that provide information about the type face, size, style and other font characteristics. Font specs should only be used with functions and commands that are designed to accept them. The format of the font spec may change in future versions.

SelectRows

Type	string	string	real	string
Description	input data	test string	field number	delimiter
Compulsory	Yes	Yes	No	No
Default			0	','

Return type: string array

Accepts an array of character delimited strings and returns an array containing a selection containing the test string at specified field. This function was developed for the parts browser mechanism but is general purpose in nature.

Example

Data input (arg 1):

```
HA-5002/HA;buf;;Buffers;;
HA-5033/HA;buf;;Buffers;;
HA5002;buf;;Buffers;;
HA5033;buf;;Buffers;;
LM6121/NS;buf;;Buffers;;1,2,4,3
MAX4178;buf_5;;Buffers;;
```

```
MAX4278;buf_5;;Buffers;;
MAX496;buf_5;;Buffers;;
```

Test string (arg 2)

```
'buf'
```

Field number (arg 3)

```
1
```

Returns:

```
HA-5002/HA;buf;;Buffers;;
HA-5033/HA;buf;;Buffers;;
HA5002;buf;;Buffers;;
HA5033;buf;;Buffers;;
LM6121/NS;buf;;Buffers;;1,2,4,3
```

SelectSimplisAnalysis

No arguments

Return type: real array

Opens SIMPLIS choose analysis dialog box. This function reads and writes the schematic's F11 window directly. The return value indicates how the user closed the box as follows:

Value	Description
0	Ok pressed
1	Cancel pressed
2	Run pressed
3	No schematic open. (Dialog doesn't open in this case)

SelGraph

No arguments

Return type: real

Returns id of selected graph. Returns 0 if no graph is open.

SelSchem

No arguments

Return type: real

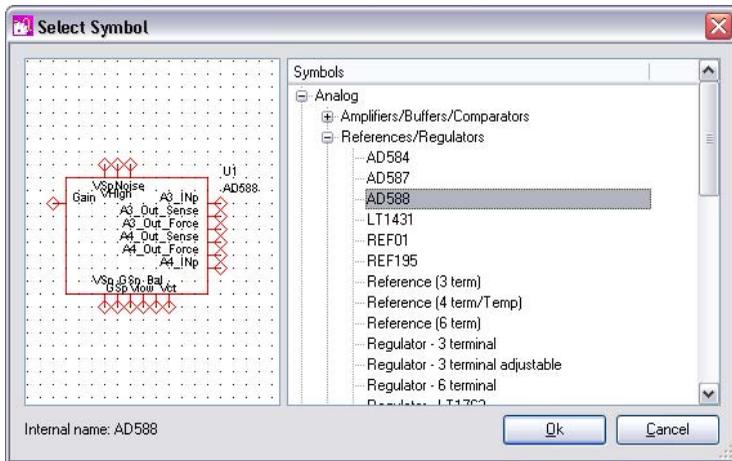
Returns 1 if at least one schematic is open otherwise 0.

SelectSymbolDialog

Type	string array	string array	string
Description	Internal symbol names	Display name and tree paths	Option
Compulsory	No	No	No
Default	Use all installed symbols	as defined by symbol	

Return type: string

Opens the following dialog box allowing the user to select a schematic symbol from the symbol library.



Argument 1

An array of internal symbol names. For the left hand graphic display to function correctly, each symbol specified must be currently installed.

Argument 2

An array of strings that describes how the symbol will be identified in the right hand pane. Expected to be a semi-colon delimited string with each token representing the node name in the tree list structure.

For example the AD588 device depicted above would be represented by the string

```
Analog;References/Regulators;AD588
```

In practice, however, it is more usual to leave this argument empty, so that the path information can be obtained from the symbol definition itself.

Argument 3

Set to 'outIndex' to change return value to an index into argument 1 instead of the actual symbol name.

Return value

The function returns the internal name of the selected symbol. If the user cancels, the function returns an empty value.

Notes

This function is used for the Place|From Symbol Library... menu. In that application, no arguments are supplied and the whole symbol library is displayed.

SetReadOnlyStatus

Type	real	real
Description	Read-only status	schematic id
Compulsory	Yes	No
Default		Current schematic

Return type: string

Sets the read-only status of the specified schematic.

Argument 1

Read only status. If 1.0, will set schematic to read-only; if 0.0 will set to writeable

Return value

Single string defining the success of the operation is defined below

Return Value	Description
'noerr'	Operation successful
'filechanged'	Set to writeable failed because the file has changed since being opened
'readonly'	The schematic's file is read-only or its security settings do not permit write operations
'locked'	The file is locked by another user
'unexpected'	An unexpected error occurred

Notes

This function sets the internal read-only status within SIMetrix and has no effect on file attributes or security settings. If a schematic is set to read-only, all attempts to perform a save operation will fail unconditionally. In addition its file will not be locked allowing other users to edit it.

When a schematic is writeable, a lock file is created which prevents other users from opening the file in writeable mode.

Shell

Type	string	string
Description	Path to executable file	options
Compulsory	Yes	No
Default		

Return type: real array

Runs an external program and returns its exit code

Argument 1

File system path to executable file. This would usually be a binary executable but may be any file that is defined as executable by the operating system.

Windows Only:

If an incomplete path is specified, the process executable will be searched in the following locations in the order given:

1. The directory where the SIMetrix binary is located
2. The current directory

3. `windows\SYSTEM32`. `windows` is the location of the Windows directory.
4. `windows\SYSTEM`
5. The windows directory
6. The directories listed in the PATH environment variable

Linux Only:

If an incomplete path is specified, the process executable will be searched in the directories specified in the PATH environment variable.

Argument 2

String array containing one or more of the options defined in the following table:

Option name	Description
'wait'	If specified, the function will not return until the called process has exited.
'command'	Windows only: Calls OS command line interpreter to execute the command supplied. This can be used to execute system commands such as 'copy' and 'move'. Ignored under Linux

Return Value

Returns a real array of length 2 as defined below:

Index	Description														
0	Process exit code. If the process is still running when this function returns, this value will be 259.														
1	Error code as follows <table border="0" style="margin-left: 20px;"> <tr> <td>0</td> <td>Process launched successfully</td> </tr> <tr> <td>1</td> <td>Command processor not found. (<i>command</i> options specified)</td> </tr> <tr> <td>2</td> <td>Cannot find file</td> </tr> <tr> <td>3</td> <td>File is not executable</td> </tr> <tr> <td>4</td> <td>Access denied</td> </tr> <tr> <td>5</td> <td>Process launch failed</td> </tr> <tr> <td>6</td> <td>Unknown failure</td> </tr> </table>	0	Process launched successfully	1	Command processor not found. (<i>command</i> options specified)	2	Cannot find file	3	File is not executable	4	Access denied	5	Process launch failed	6	Unknown failure
0	Process launched successfully														
1	Command processor not found. (<i>command</i> options specified)														
2	Cannot find file														
3	File is not executable														
4	Access denied														
5	Process launch failed														
6	Unknown failure														

ShellExecute

Type	string	string	string	string
Description	File	parameters	default directory	verb
Compulsory	Yes	No	No	No
Default		none	current directory	'open'

Return type: string

This function performs no action on Linux and just returns 'NotImplemented'

Performs an operation on a windows registered file. The operation to be performed is determined by how the file is associated by the system. For example, if the file has the extension PDF, the Adobe Acrobat or Adobe Acrobat Reader would be started to open the file. (Assuming Acrobat is installed and correctly associated)

Argument 1

Name of file to process. This can also be the path to a directory, in which case an 'explorer' window will be opened.

Argument 2

Parameters to be passed if the file is an executable process. This should be empty if arg 1 is a document file.

Arguments 3

Default directory for application that processes the file

Argument 4

'Verb' that defines the operation to be performed. This would usually be 'open' but could be 'print' or any other operation that is defined for that type of file.

Return Value

Returns one of the following

Value	Description
'OK'	Function completed successfully
'NotFound'	File not found
'BadFormat'	File format was incorrect
'AccessDenied'	File could not be accessed due to insufficient privilege

Value	Description
'NoAssoc'	File has no association for specified verb
'Share'	File could not be accessed because of a sharing violation
'Other'	Function failed for other reason
'NotImplemented'	Function not implemented on this platform. This is what is always returned on Linux versions.

sign

Type	real array
Description	vector
Compulsory	Yes
Default	

Return type: real array

Returns 1 if argument is greater than 0 otherwise returns 0.

SimetrixFileInfo

Type	string
Description	File name
Compulsory	Yes
Default	

Return type: string array

Returns information about a SIMetrix file. Currently this function will only return information about version 4.1 or later schematic files.

Return Value

Return value will be an array of length 3. The first element will currently be one of the values, 'Schematic', 'Unknown' or 'CantOpen'. The second element reports the file format version. The third element will be one of:

Value	Description
'Schematic'	File is SIMetrix component or schematic file and contains just a schematic. (4.1 or later)
'Symbol'	File is a SIMetrix component file and contains only the symbol part of the component
'Symbol Schematic'	File is a SIMetrix component file and contains both the symbol part and the schematic part of the component

SimulationHasErrors

No arguments

Return type: real

Return 1 if the most recent simulation failed with an error. Otherwise returns 0.

Note that the function will return 0 if no simulation has been run or if the simulator has been reset using the Reset command. It will also return 0, if the simulation failed because of a fatal error that caused the simulator process to restart. This occurs when an access violation or floating point exception occurs.

sin

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Return sine of argument specified in radians.

sin_deg

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Return sine of argument specified in degrees.

Sleep

Type	real
Description	Time in seconds
Compulsory	Yes
Default	

Return type: real

Executes a timed delay.

Argument 1

Delay in seconds. The function has a resolution of 100mS and so the delay will be integral multiples of that amount.

Return Value

Function returns the value of the argument.

Sort

Type	string	string
Description	string data	Options
Compulsory	Yes	No
Default		

Return type: string array

Performs alphanumeric sort on string array.

Argument 1

String array to be sorted

Argument 2

May be set to 'unique' in which case any duplicates in argument 1 will be eliminated.

Return Value

Result is string array containing the contents of argument 1 sorted in alphanumeric order.

SortIdx

Type	any array	string
Description	items to sort	sort direction
Compulsory	Yes	No
Default		'forward'

Return type: real array

Sorts the items in argument 1 but instead of returning the actual sorted data the function returns the indexes of the sorted values into the original array. The method of sorting depends on the data type as follows:

string	Alphabetic
real	Numeric
complex	Numeric - uses magnitude

If the second arguments is 'reverse' the sort is performed in reverse order.

SourceDialog

Type	string
Description	initialisation string
Compulsory	No
Default	<<empty>>

Return type: string

This is a special purpose function used to select a voltage or current signal source. It opens a dialog box whose controls are initialised according to the string passed as the

function's arguments. It returns a string giving the definition of the source selected by the user. The string may be used as the value for a current or voltage source.

SplitPath

Type	string
Description	path
Compulsory	Yes
Default	

Return type: string array

Splits file system pathname into its component path. Return value is string array of length 4:

0	Drive including ':'. E.g. 'C:'
1	Directory including prefix and postfix '\'. E.g. "Program Files\SIMatrix\'
2	Filename without extension. E.g. 'SIMatrix'
3	Extension including period. E.g. '.EXE'

SprintfNumber

Type	string	real ...
Description	format	Arguments 1-8: values
Compulsory	Yes	No
Default		

Return type: string

Returns a string formatted according to a format specification.

Argument 1

Format specification. The format used is essentially the same as that used for the 'printf' range of functions provided in the 'C' programming language. However, only real arguments are supported and so only format types %e, %E, %f, %g and %G are supported.

Arguments 2-8

Values used for '%' format specs in the format string.

Return Value

Formatted string

sqrt

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Returns the square root of the argument. If the argument is real and negative, an error will result. If however the argument is complex a complex result will be returned.

Str

Type	any
Description	
Compulsory	Yes
Default	

Return type: string

Returns the argument converted to a string.

StringLength

Type	string
Description	input string
Compulsory	Yes
Default	

Return type: real

Returns the number of characters in the supplied string.

StrStr

Type	string	string	real
Description	input string	sub string	offset
Compulsory	Yes	Yes	No
Default			0

Return type: real

Locates the sub string in argument 2 in the input string. If found the function will return the character offset of the sub string. If not found the function will return -1.

Argument 1

String to search

Argument 2

Sub-string

Argument 3

Offset into search string where search should begin.

Return Value

Number of characters from start of search string where sub string starts. -1 if substring is not found.

SubstChar

Type	string	string	string
Description	string to process	characters to replace	character to substitute
Compulsory	Yes	Yes	Yes
Default			

Return type: string

Scans string in arg 1 and replaces characters found in arg 2 with the character specified in arg 3. Returns the result.

SubstString

Type	string	string	string	string
Description	string to process	Search string	Substitute string	Options
Compulsory	Yes	Yes	Yes	No
Default				

Return type: string

Replaces a substring in a string.

Argument 1

Input string.

Argument 2

Substring searched in input string.

Argument 3

The substring defined in argument 2 found in the input string is replaced with this value. If arg 4 is set to 'all' all substrings found will be replaced, otherwise only the first will be replaced.

Argument 4

Options. If set to 'all' all substrings located in the string will be replaced. Otherwise, only the first occurrence will be replaced.

Return value

Result of string substitution. Note that only the first occurrence of the substring is replaced.

SumNoise

Type	real	real	real
Description	vector	start x value	end x value
Compulsory	Yes	No	No
Default		start of input vector	end of input vector

Return type: real array

Identical to RootSumOfSquares function. See [page 277](#).

SymbolGen

Type	string array	string	string	string
Description	Definition of existing symbols in "Compact Text Format"	Name of initial symbol	Catalog for new symbol	Initial property style
Compulsory	Yes	No	No	No
Default		none		

Return type: string array

Special purpose function used to create new schematic symbols. Opens the "Symbol Generator" dialog box. If the user selects "OK" the new symbol will be added to the global library currently in memory and the function will return the name and description of the new symbol in elements 1 and 2 respectively of the returned value. If the user selects "Cancel" the function will return empty data.

Argument 1

Passes existing symbol definitions in "compact text format" described on [page 202](#).

Argument 2

Name of symbol initially selected.

Argument 3

Catalog for initial symbol

Argument 4

When a new symbol is created by the user by pressing the New... button the property sheet is initialised according to the setting of this argument defined as follows:

'block'	Model and Ref properties defined
'symbol'	Value, Model and Ref defined.
<<empty>>	No properties defined

SymbolInfoDialog

Type	string	string
Description	initial settings	available catalogs
Compulsory	No	No
Default		

Return type: string array

Opens a dialog box allowing the specification of symbol details.

Argument 1

String array length 3 specifying initial settings.

0	Symbol name
1	Display name
2	Catalog
3	Path
4	If 'component', save as component initially selected
5	If '1' "All references to symbol automatically updated" box will be checked.

Argument 2

List of available catalogs entered into catalog list box.

Return value

String array of length 4 as follows

0	Symbol name entry
1	Display name entry
2	Catalog selected
3	'Save to' radio button: 1 Global library 2 Current schematic only 3 Both
4	File path
5	'1' if 'All references to symbol automatically updated' box is checked, otherwise '0'

SymbolLibraryManagerDialog

No arguments

Return type: string array

Opens the Symbol Library Manager dialog box. See the Schematic Editor chapter of the *User's Manual* for details of this feature.

Return Value

Index	Description
0	User operation: 0 Close button pressed 1 Place button pressed 2 Edit button pressed
1	Internal name of selected symbol
2	Full path of selected library file
3	Empty - reserved for future use.

SymbolName

Type	string	string	real
Description	property name	property value	Schematic handle
Compulsory	Yes	Yes	No
Default			-1

Return type: string

Argument 1, 2

Property name and value to identify instance. If these arguments are not supplied, the selected instance, if any, will be used instead. If there are no selected instances or no instances that match the arguments, the function will return an empty vector. If the arguments identify more than one instance, the function will return information for one of them but there are no rules to define which one.

Argument 3

Schematic handle as returned by the `OpenSchematic` function ([page 249](#)). This allows this function to be used with a schematic that is not open or not currently selected. If omitted or -1, the currently selected schematic will be used.

Return Value

Returns the symbol name used by the instance defined by property name and value supplied in arguments 1 and 2.

SymbolNames

Type	real	string	string
Description	Schematic handle	Property name	Property value
Compulsory	No	No	No
Default	-1	Use selected	Use all with property name in arg 2

Return type: string array

Returns symbol names of schematic instances.

Argument 1

Schematic handle as returned by the `OpenSchematic` function ([page 249](#)). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

Argument 2,3

If present these arguments identify the instances to be examined. If only argument 2 is specified then all instances on the specified schematic that possess that property will be used. If argument 3 is also present then the instance name and value must match argument 2 and 3 respectively. If neither are present the selected instances will be used.

Return Value

String array containing the symbol names for the instances identified by this functions arguments.

Note that this function complements `PropValues2` ([page 261](#)) and `PropFlags2` ([page 258](#)) and will return the same number of values and in the same order as those function given the same arguments.

SymbolPinOrder

Type	string array
Description	new pin order
Compulsory	No
Default	

Return type: string array

Returns pin order of symbol in currently open symbol editor sheet. Also sets new pin order if argument supplied.

Argument 1

Array of strings with names of pins in the required order.

Return value

Array of strings containing pin names of current symbol in the current order. If no symbol editor sheets are open, the function returns an empty vector.

SystemValue

***** UNSUPPORTED FUNCTION ***** see [page 72](#)

Type	string
Description	Value name
Compulsory	Yes
Default	

Return type: string

Returns the value of a system defined variable. System defined variables are values that are 'hard-wired' in the program. This function provides access to these variables. The function is used by some internal scripts.

TableDialog

Type	real array	string array
Description	Geometry	Cell initial values
Compulsory	Yes	No
Default		

Return type: string array

Displays a spreadsheet style table to allow the user to enter tabular data. See example below for a picture.

Argument 1

Real array of length 2. First element is the number of rows initially displayed and the second element is the number of columns. Note that these are just the initial values. The user may subsequently add or delete rows and columns.

Argument 2

An array of strings to define the initial cell entries. If not supplied, the cells will begin empty.

Each element in the array is a semi-colon delimited string and defines a complete row. The cell entries are sequentially loaded from the delimited fields in each row.

Return value

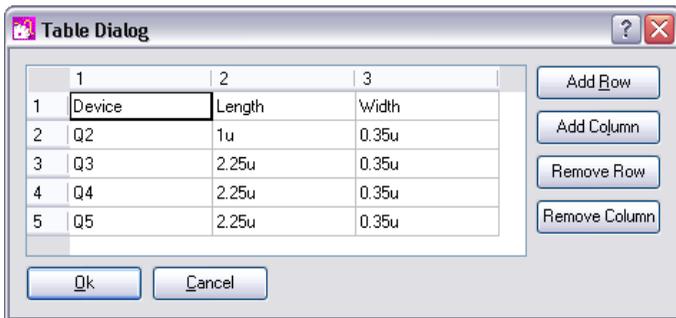
Return value will be in the same format at argument 2 and provide the contents of the cells as entered by the user.

Example

A call to:

```
TableDialog ([5,3], ["'Device;Length;Width;', 'Q2;1u;0.35u;',  
'Q3;2.25u;0.35u;', 'Q4;2.25u;0.35u;', 'Q5;2.25u;0.35u;'])
```

will show this dialog:



tan

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Return tan of argument specified in radians.

tan_deg

Type	real/complex array
Description	vector
Compulsory	Yes
Default	

Return type: real/complex array

Return tan of argument specified in degrees.

TemplateGetPropValue

Type	string	string
Description	REF property value	Property name
Compulsory	Yes	Yes
Default		

Return type: string

This function may only be used in Template scripts. These are used for advanced netlist customisation. See [“Schematic Template Scripts” on page 464](#) for more details.

Function returns the value of the property defined in argument 2 for the schematic instance defined by the REF property value given in argument 1.

TemplateResolve

Type	string	string
Description	REF property value	Template value
Compulsory	Yes	Yes
Default		

Return type: string

This function may only be used in Template scripts. These are used for advanced netlist customisation. See “[Schematic Template Scripts](#)” on page 464 for more details.

Function processes argument 2 as if it were a TEMPLATE property for the instance defined by argument 1. The return value is what the template resolves to.

Time

Type	string
Description	option
Compulsory	No
Default	<<empty>>

Return type: string

Returns the current time in the format specified in control panel.

TransformerDialog

Type	string	string	real
Description	Array of Core Materials	Array of core parts	Initialisation data
Compulsory	Yes	Yes	Yes
Default			

Return type: real array

Argument 3

1	Material index (arg 1) (element 2 = -1)
2	Core part index (arg2).
3	Number of primaries
4	Number of secondaries
5	Effective area (element 2 = -1)
6	Effective length (element 2 = -1)
7	Effective relative permeability (element 2 = -1)
8	Number of turns, winding 1
9	Number of turns, winding 2
10	Coupling coefficient winding 1 - 2
11	Number of turns, winding 3
12	Coupling factor, winding 1 - 3
13	Coupling factor, winding 2 - 3

Return value elements

1	Material index (arg 1) (element 2 = -1)
2	Core part index (arg 2).
3	Effective area (element 2 = -1)
4	Effective length (element 2 = -1)
5	Effective relative permeability (element 2 = -1)
6	Number of turns, winding 1
7	Number of turns, winding 2
8	Coupling coefficient winding 1 - 2
9	Number of turns, winding 3
10	Coupling factor, winding 1 - 3
11	Coupling factor, winding 2 - 3

Special purpose function used for selection of non linear magnetic components. Opens 1 of three styles of dialog box depending on the winding configuration. The user can either select a standard core configuration or define custom core parameters. In the latter case, the Core part index will be -1 otherwise an index into the array specified in argument 2 will be returned. The same rules apply to the initialisation data supplied in argument 3.

TranslateLogicalPath

Type	string
Description	Symbolic path
Compulsory	Yes
Default	

Return type: string

Converts symbolic path to a physical path.

Argument 1

Symbolic path as described in the “Sundry Topics” chapter of the *User's Manual*.

Return Value

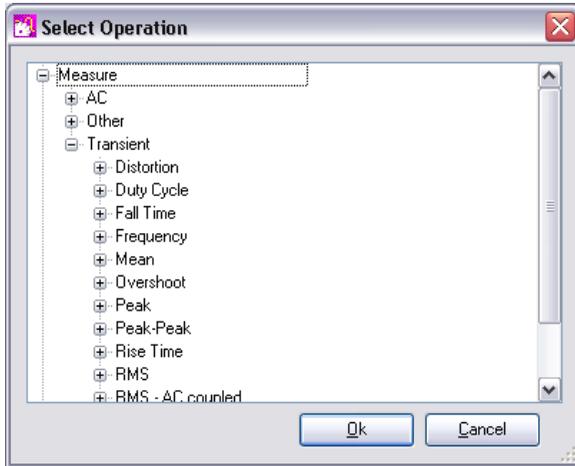
Returns actual file system path.

TreeListDialog

Type	string array	string array
Description	strings	options
Compulsory	Yes	No
Default		['Select Item', ", '0', 'sort', 'false']

Return type: real

Opens the following dialog box allowing the user to specify an item in tree structured list.



Argument 1

Specifies the items to be displayed in the tree list. These are arranged in semi-colon delimited fields with each field specifying a “branch” of the tree. For example, in the above diagram, the item shown as “Full” would be specified as an element of argument 1 as “Measure;Transient;RMS;Full”.

Argument 2

An array of strings of max length 5 specifying various other characteristics as defined below:

0	Dialog caption
1	Identifies an item to be initially selected using the same format as the entries in argument 1.
2	Initial expand level. '0' for no expansion, '1' expands first level of tree etc.
3	Items will be alphabetically sorted <i>unless</i> this is set to 'nosort'
4	Items may selected and the box closed by double clicking <i>unless</i> this item is set to 'true'

Return value

Returns index into argument 1 of selected item. If no item is selected, the function returns -1. If the user selects Cancel the function returns an empty vector.

TRUE

Type	string	string
Description	vector name	option
Compulsory	Yes	No
Default		<<empty>>

Return type: real

Returns TRUE (1) if the vector specified by name in argument 1 exists AND is non-zero. If argument 2 is set to 'SearchCurrent', the *current* group as well as the *local* and *global* groups will be searched for the vector, otherwise only the *local* and *global* groups will be searched. See “Groups” on page 37 for an explanation of groups.

Truncate

Type	real array	real	real
Description	vector	start x value	end x value
Compulsory	Yes	No	No
Default		start of vector	end of vector

Return type: real array

Returns a portion of the input vector with defined start and end points. Interpolation will be used to create the first and last points of the result if the start and end values do not coincide with actual points in the input vector.

Arguments 2 and 3 define the beginning and end of the vector.

Example

Suppose we have a vector called VOUT which was the result of a simulation running from 0 to 1mS. We want to perform some analysis on a portion of it from 250µS to 750µS. The following call to Truncate would do this:

```
Truncate(VOUT, 250u, 750u)
```

If VOUT did not actually have points at 250µS and 750µS then the function would create them by interpolation. Note that the function will not extrapolate points before the start or after the end of the input vector.

Units

Type	any
Description	vector or vector name
Compulsory	Yes
Default	

Return type: string

Returns the physical units of the argument. Possible return values are

" (meaning dimensionless)

'?' (meaning unknown)

'V'

'A'

'Secs'

'Hertz'

'Ohm'

'Sie'

'F'

'H'

'J'

'W'

'C'

'Vs'

'V^2'

'V^2/Hz'

'V/rtHz'

'A^2'

'A^2/Hz'

'A/rtHz'

'V/s'

See also

PhysType [page 254](#)

unitvec

Type	real
Description	Number of elements in result
Compulsory	Yes
Default	

Return type: real array

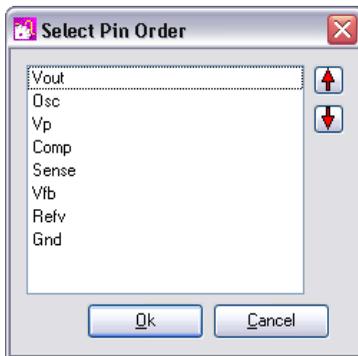
Returns a vector consisting of all 1's. Argument specifies length of vector.

UpDownDialog

Type	string array	string
Description	strings to sort	box caption
Compulsory	Yes	No
Default		'Select Item Order'

Return type: string array

Opens the following dialog box to allow the user to rearrange the order of a list of strings.



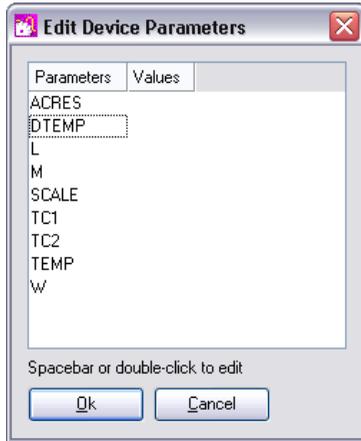
The box displays the strings given in argument 1 in the order supplied. The user can rearrange these using the up and down arrow buttons. When the user presses OK the function return the strings in the new order. If the user cancels the box the function returns an empty vector.

UserParametersDialog

Type	string array	string array	string
Description	Names	Values	Title
Compulsory	Yes	Yes	No
Default			'Edit Device Parameters'

Return type: string array

Opens the following dialog box and enters the names and values specified in the arguments.



The user may edit any of the values by double clicking an entry or pressing the space bar. The function returns a string array holding the new values for each parameter.

Val

Type	any
Description	input value
Compulsory	Yes
Default	

Return type: real/complex

Returns argument converted to a value. The conversion assumes that the string supplied is an expression.

See also

[Str\(\)](#) [page 298](#)

ValueDialog

Type	real	string	string	string
Description	Initial edit control values	Edit control labels	Dialog box caption	Special characteristics
Compulsory	No	No	No	No
Default	1	empty	empty	none

Return type: real array

Opens a dialog box with up to 10 edit controls allowing numeric values to be entered.

The number of edit controls displayed is determined by the length of the first argument. If this is omitted, all 10 will be displayed. Argument 1 specifies the initial values set in each of the controls.

Argument 2 supplies the text of the label displayed to the left of each edit control. The width of the dialog box will be adjusted to accommodate the length of this text.

Argument 3 specifies the text in the title bar of the dialog box

Argument 4 attaches special characteristics for particular applications. The value of this argument and meaning is as follows:

Value	Action
'Switch'	For use to specify VC switches. Assumes box 1 is for 'On resistance' and box 2 for 'Off resistance'. Action is modified to ensure 'On resistance' < 'Off resistance'
'Transformer'	For use to specify ideal transformers. Assumes box 1 is 'Turns ratio', box 2 'Primary Inductance' and box 3 is 'Coupling Factor' Hides up-down control for box 3. Min values for boxes 1 and 2 set to 1e-18 Box 3 range 0 to 0.999999
'TransmissionLine'	For use to specify lossless transmission lines Assumes box 1 is 'Characteristic Impedance' and box 2 is 'Delay'. Sets box 1 minimum value to 1e-18 and box 2 minimum value to 1e-21

Any other value supplied for argument 4 will be treated as the default. In this case all boxes are allowed to vary over a range of -1e18 to +1e18.

The function returns an array representing the user selected value in each box. If cancelled it returns an empty vector.

See also

[NewValueDialog \(page 244\)](#)

[BoolSelect \(page 79\)](#)

[EditSelect \(page 128\)](#)

[RadioSelect \(page 265\)](#)

Vec

Type	string	string
Description	Vector name	Group name
Compulsory	Yes	No
Default		Current group

Return type: depends on arg 1

Returns the data for the vector specified by the arguments.

The purpose of this function is to provide a means of obtaining the data for vectors whose names violate vector name rules. Such vectors can be generated by the simulator if there are - for example - net names containing arithmetic characters. The simulator will create a vector of the same name but because the vector name contains an arithmetic character it is not possible to access the vector's data by the normal method.

For example, suppose a simulation was run on a circuit that contains a net called "IN+". A vector will be created called IN+. If the command to plot this vector were executed - "Plot IN+" - an error would result because "IN+" is an incomplete arithmetic expression. Instead the following can be used:

```
Plot Vec('IN+')
```

The schematic cross-probing mechanism will automatically use this syntax when needed.

vector

Type	real
Description	Number of elements in result
Compulsory	Yes
Default	

Return type: real array

Returns a vector with length specified by the argument. The value in each element of the vector equals its index.

See also

UnitVec [page 313](#)

VectorsInGroup

Type	string	string array
Description	group name	options
Compulsory	No	No
Default	Current group	

Return type: string array

Returns the names or optionally the physical type of all vectors in the specified group. Argument 2 is a string array that may contain values of 'PhysType' and/or 'RealOnly'. If 'PhysType' is present the physical type (e.g. 'voltage', 'current', 'time' etc.) of the vectors will be returned otherwise the function will return their names. If 'RealOnly' is present, only values of type 'Real' will be returned. Complex values, string values and aliases values will be excluded.

VersionInfo

No arguments

Return type: string array

Returns a string array of length 7 defined as follows:

0	Product name. E.g. "SIMetrix Micron AD"
1	Major Version number (3.1, 4.0 etc.)
2	Maintenance version. (empty or a single letter)

3	Internal product name. (E.g. "Micron A-D")
4	Feature string allowing script to determine available functionality. This will be a combination of the following separated by the ' ' character: Basic Always present AD Digital simulator enabled Micron CMOS device models enabled Schematic Schematic enabled Advanced Advanced analysis modes enabled Scripts Scripting enabled Rtn Real time noise enabled Simplis_If SIMPLIS simulator interface present
5	Full version string - usually element 1 and 2 concatenated
6	Base product name

WirePoints

Type	string	string
Description	wire handle	Schematic Handle
Compulsory	Yes	No
Default		-1

Return type: real array

Argument 1

Handle of schematic wire segment. Wire handles are returned by the functions [Wires \(page 320\)](#), [NetWires \(page 242\)](#) and [SelectedWires \(page 285\)](#).

Argument 2

Schematic handle as returned by the [OpenSchematic \(page 249\)](#) function. This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

Return Value

Returns a numeric vector of length 4 providing the sheet locations of the each termination of the specified wire.

The four values in the vector are defined in the following table,

Index	Description
0	X-co-ordinate for termination 1
1	Y-co-ordinate for termination 1
2	X-co-ordinate for termination 2
3	Y-co-ordinate for termination 2

The functions returns an empty vector ([page 28](#)) if the wire handle supplied is invalid.

See also

InstPoints [page 220](#)

Wires

Type	real
Description	Schematic handle
Compulsory	No
Default	-1

Return type: string array

Returns array of strings holding handles for all wires in the specified schematic. Wire handles are used by the function WirePoints ([page 319](#)) and the commands Select ([page 419](#)) and SetHighlight ([page 421](#))

Argument 1

Schematic handle as returned by the OpenSchematic function ([page 249](#)). This allows this function to be used with a schematic that is not open or not currently selected. If equal to -1, the currently selected schematic will be used.

See also

NetWires [page 242](#)

SelectedWires [page 285](#)

WriteConfigSetting

Type	string	string	string
Description	Section	Key	Value
Compulsory	Yes	Yes	No
Default			

Return type: real

Writes a configuration setting. Configuration settings are stored in the configuration file. See ‘Configuration Settings’ in Chapter 13 of the *User’s Manual* for more information. Settings are defined by a key-value pair and are arranged into sections. The function writes the value in argument three to the specified key and section. If the value is missing, the setting will be deleted.

Argument 1

Section name

Argument 2

Key name

Argument 3

Value to set. Setting will be deleted if this is omitted.

Return Value

Returns TRUE if value was successfully written. Otherwise returns FALSE.

See Also

[ReadConfigSetting \(page 267\)](#)

WriteF11Lines

Type	string array
Description	Lines
Compulsory	Yes
Default	

Return type: real

Writes lines directly to the F11 window overwriting any existing lines.

Argument 1

Lines to write in the form of a string array. Each element in the array creates a new line.

Return Value

Returns 1.0 if the function is successful otherwise returns 0.0. The function will only fail if there are no schematics open.

See Also

[AppendTextWindow \(page 348\)](#)
[GetF11Lines \(page 160\)](#)

WriteIniKey

Type	string	string	string	string
Description	File	Section	Key	Value
Compulsory	Yes	Yes	Yes	No
Default				Empty string

Return type: real

Writes a value to an 'INI' file. See [ReadIniKey \(page 269\)](#) for more information on INI files.

Argument 1

File name. You should always supply a full path for this argument. If you supply just a file name, the system will assume that the file is in the WINDOWS directory. This behaviour may be changed in future versions. For maximum future compatibility, always use a full path.

Argument 2

Section name.

Argument 3

Key name.

Argument 4

Key value

Return Value

Returns 1 if function successful. Otherwise returns 0.

WriteRawData

Type	real/complex array	string	string	string
Description	data	File name	Options	Format of index display
Compulsory	Yes	Yes	No	No
Default				'%d'

Return type: string

Writes data to the specified file in a SPICE3 raw file compatible format. See the built in script `write_raw_file` for an application example. This can be found on the install CD.

The function returns a single string according to the success or otherwise of the operation. Possible values are: 'success', 'nodata' and 'fileopenfail'.

WriteRegSetting

Type	string	string	string	string
Description	Key path	Value name	Value to be written	Top level tree
Compulsory	Yes	Yes	Yes	No
Default				'HKCU'

Return type: string

Writes a string value to the windows registry.

On Linux the registry is simulated using settings in the SIMetrix configuration file.

Argument 1

Name of key. This must be a full path from the top level. E.g. 'Software\SIMetrix\Version42\Options'

Argument 2

Name of value to be read

Argument 3

Value to be written to key

Argument 4

Top level tree. This may be either 'HKEY_CURRENT_USER' or 'HKEY_LOCAL_MACHINE' or their respective abbreviations HKCU and HKLM. Note that you must have administrator rights to write to the HKEY_LOCAL_MACHINE tree.

Return Value

Returns one of three string values as defined below:

Value	Meaning
'Ok'	Function executed successfully
'WriteFailed'	Could not write that value
'InvalidTreeName'	Arg 4 invalid.

WriteSchemProp

Type	string	string	string
Description	property name	property value	option
Compulsory	Yes	Yes	No
Default			

Return type: real

Writes a schematic window property. If argument 3 is set to 'Create' the function will create the property if it doesn't already exist, otherwise the function can only change the value of an existing property. There are three writeable properties that are built-in, namely 'RootPath', 'Reference' and 'UserStatus'. See ReadSchemProp - [page 270](#) - for details.

The function returns an integer that indicates the success of the operation as follows:

-1	No schematic windows open
0	Success
1	Property does not exist and 'Create' not specified
2	Property is read only. (e.g. the 'Path' property)
3	Property successfully created

XCursor

No arguments

Return type: real

Returns the horizontal position of the graph measurement cursor. If there is no graph open or cursors are not enabled, the function returns 0.

XDatum

No arguments

Return type: real

Returns the horizontal position of the graph reference cursor. If there is no graph open or cursors are not enabled, the function returns 0.

XFromY

Type	real	real	real	real
Description	input vector	Y value	Interpolation order (1 or 2)	Direction
Compulsory	Yes	Yes	No	No
Default			2	0

Return type: real array

Returns an array of values specifying the horizontal location(s) where the specified vector (argument 1) crosses the given y value (argument 2). If the vector never crosses the given value, an empty result is returned. The sampled input vector is interpolated to produce the final result. Interpolation order is specified by argument 3.

Argument 4 specifies edge direction. If set to 0 either direction will be accepted. If set to 1 only positive edges will be detected and if set to -1 only negative edges will be detected.

Note that unlike other functions that use interpolation, XFromY can only use an interpolation order of 1 or 2. If a value larger than 2 is specified, 2 will be assumed.

XY

Type	real array	real array
Description	y vector	x vector
Compulsory	Yes	Yes
Default		

Return type: real array

Creates an *XY Vector* from two separate vectors. An *XY Vector* is a vector that has a reference (see “[Vector References](#)” on page 42). The resulting vector will have y values defined by argument 1 and the x values (i.e. its reference) of argument 2.

YCursor

No arguments

Return type: real

Returns the vertical position of the graph measurement cursor. If there is no graph open or cursors are not enabled, the function returns 0.

YDatum

No arguments

Return type: real

Returns the vertical position of the graph reference cursor. If there is no graph open or cursors are not enabled, the function returns 0.

YFromX

Type	real	real	real
Description	input vector	X value	Interpolation order (1 or greater)
Compulsory	Yes	Yes	No
Default			2

Return type: real array

Returns an array of values (usually a single value) specifying the vertical value of the specified vector (argument 1) at the given x value (argument 2). If the given x-value is out of range an empty result (see [page 28](#)) is returned. The sampled input vector is interpolated to produce the final result. Interpolation order is specified by argument 3.

Chapter 5 Command Summary

Notation

Symbols used

Square brackets: []

These signify a command line parameter or switch which is optional.

Pipe symbol: |

This signifies either/or.

Ellipsis: . . .

This signifies 1 or more optional multiple entries.

Fonts

Anything that would be typed in is displayed in a *fixed width font*.

Command line parameters are in *italics*.

Case

Although upper and lower cases are used for the command names, they are NOT in fact case sensitive.

Examples

Example 1

```
OpenGroup [ /text ] [ filename ]
```

Both */text* (a switch) and *filename* (a parameter) are optional in the above example.

So the following are all legitimate:

```
OpenGroup
OpenGroup /text
OpenGroup run23.dat
OpenGroup /text output.txt
```

Example 2

```
DelCrv curve_number...
```

1 or more *curve_number* parameters may be given.

So the following are all legitimate:

```
DelCrv 1 2 3
DelCrv 1
```

Command Summary

The following table lists all commands available. Shaded boxes indicate commands that are either new for version 5.5, have been updated or their documentation has been updated.

Abort	Aborts the current simulation
AbortSIMPLIS	Sends a signal to the SIMPLIS simulator instructing it to abort
About	Displays the <i>about box</i>
AddArc	Symbol Definition Command. Create whole circles and ellipses as well as arcs of circles and ellipses
AddCirc	Symbol Definition Command. Creates a circle
AddCurveMarker	Adds a curve marker to the currently selected graph sheet
AddFreeText	Adds a free text item to the currently selected graph sheet
AddGraphDimension	Adds a dimension object to a graph
AddLegend	Adds a legend box to the currently selected graph
AddLegendProp	Adds a property to a graph legend
AddPin	Symbol Definition Command. Adds a pin to a symbol
AddProp	Symbol Definition Command. Adds a property to a symbol definition
AddSeg	Symbol Definition Command. Adds a line segment to a symbol
AddSymbolProperty	Adds a property to the symbol currently open in the symbol editor
AddTextBox	Adds a Text Box to the currently selected graph
Anno	Annotate schematic with unique component references
AppendTextWindow	Append text to simulator command (F11) window
Arguments	Declare arguments to script
BuildDefaultOptions	Resets preference settings to factory defaults
Cancel	Cancel schematic interactive command
Cd	Change current working directory
ChangeArcAttributes	Modify symbol arc attributes
ChangeSymbolProperty	Modify property value/attributes in symbol editor
ClearMessageWindow	Clears the command shell message window
Close	Close schematic or graph window

CloseGraphSheet	Closes the current tabbed sheet in the selected graph window
ClosePrinter	Conclude print job
CloseSheet	Close tabbed sheet in schematic editor
CloseSimplisStatusBox	Closes the SIMPLIS simulation status box
CollectGarbage	Deletes temporary vectors
CompareSymbolLibs	Compares two symbol libraries
Copy	Copy selected schematic components then paste (Interactive)
CopyClipGraph	Copy graph to clipboard to paste to other applications
CopyClipSchem	Copy schematic to clipboard to paste to other applications
CopyFile	Copy a file
CopyLocalSymbol	Copy local symbol to global library
CreateFont	Create a named font object
CreateGroup	Creates a data group
CreateSym	Symbol definition: Start definition
CreateToolBar	Creates a new empty toolbar
CreateToolButton	Creates or redefines a tool bar button
CursorMode	Enable/disable/step graph cursors
Curve	Create new curve in existing graph
CurveEditCopy	Copy specified curves to the internal clipboard
DefButton	Defines the command executed when a button is pressed
DefineToolBar	Defines the action for a schematic button
DefKey	Define keyboard key.
DefMenu	Define fixed or popup menu item
Del	Delete file
DelCrv	Delete curve
Delete	Delete selected schematic items
DeleteAxis	Delete specified y-axis or grid
DeleteGraphAnno	Delete graph annotation object
DeleteSymbolProperty	Delete property in symbol editor
DelGroup	Delete group (of simulation data)
DelLegendProp	Delete graph legend property

DelMenu	Delete menu item
DelProp	Delete schematic instance property
DelSym	Delete symbol definition
DelSymLib	Delete entire symbol library
Detach	Detach selected wires. (Disables rubberbanding)
Discard	Free up memory used by vectors
Display	Display variables in current group
DrawArc	Initiate arc drawing mode in symbol editor
DrawPin	Initiate pin placement mode in symbol editor
Echo	Display text in message window or write text to file
EditColour	Edit a colour
EditCopy	Copy selected schematic items to clipboard for pasting to other schematics or other applications.
EditCut	Deletes selected components and places them in the clipboard
EditFile	Edit text file
EditFont	Edit a font
EditPaste	Paste clipboard data to schematic. (Interactive)
EditPin	Edit a pin name of a symbol in the currently installed symbol library
EndSym	Symbol definition: terminate definition
Execute	Execute script
Focus	Focus a window
FocusShell	Selects the Command Shell and assigns it keyboard focus
GraphZoomMode	Select mode for next cursor zoom function
Help	Display help system.
HideCurve	Hide specified curve
HighlightCurve	Highlights the selected curve
Hint	Display a hint to the user
HourGlass	Displays the hourglass cursor shape indicating that some action is in progress
ImportSymbol	Import symbol to symbol editor
Inst	Place component on schematic. (Interactive unless /loc specified)
KeepGroup	Prevent group (simulation data) from being automatically deleted.

Let	Evaluate expression
Listing	Display or write to file current netlist.
ListModels	Create dictionary of currently installed models
ListOptions	List all global options to file
ListStdButtonDefs	Lists the built in toolbar button definitions
ListStdKeys	Write standard key definitions to file
LoadModelIndex	Re load model library indexes into memory
MakeAlias	Make alias variable
MakeCatalog	Makes OUT.CAT file for use by parts browser
MakeSymbolScript	Build script for symbol(s)
MakeTree	Creates the specified directory path
MCD	Make and change current working directory
MD	Make directory
Message	Display message in schematic status window
MessageBox	Displays message box
Move	Move selected schematic items (Interactive)
MoveCurve	Move specified curve to new axis
MoveFile	Moves a file to a new location
MoveMenu	Moves the position of a menu item by a specified count
MoveProperty	Move a property on a schematic instance
Netlist	Create netlist of current schematic
NewAxis	Create new y-axis
NewGraphWindow	Open new graph window
NewGrid	Create new graph grid
NewPrinterPage	Start new page in print job
NewSchem	Open new schematic window
NewSymbol	Open new symbol sheet in symbol editor
NoPaint	Disable graph painting for duration of current script
NoUndo	Inhibits saving to undo buffer
OpenGraph	Opens a SIMetrix graph file
OpenGroup	Create new group (of simulation data) from data file.
OpenPrinter	Begin print job
OpenRawFile	Opens a SPICE 3 format ASCII raw file.

OpenSchem	Open existing schematic
OpenSimplisStatusBox	Opens the SIMPLIS simulation status box
OptionsDialog	Open options dialog box
Pan	Pan (scroll) schematic specified number of grid squares
Pause	Pause current simulation
PinDef	Create new pin for generated symbol
PlaceCursor	Position graph cursor
Plot	Create new graph window and plot curve
PreProcessNetlist	Pre-processes a netlist. Intended for use with SIMPLIS but is general purpose in nature
PrintGraph	Print graph. (Interactive)
PrintSchematic	Print current schematic in non-interactive print job
Probe	Change schematic cursor to probe and wait for mouse click. (Interactive)
Prop	Change/add property of/to schematic instance
Protect	Protect selected schematic components
Quit	Exit SIMetrix
RD	Remove directory
ReadLogicCompatibility	Read logic compatibility tables
RebuildSymbols	Reload symbols from library file
Redirect	Redirect messages to message window to file
RedirectMessages	Redirects all command shell messages to a file
Redo	Re do last undo operation
RegisterUserFunction	Register a user defined function
RenameLibs	Run rename model utility
RepeatLastMenu	Executes most recently selected the menu
Reset	Release memory used for simulation
RestartTran	Restart a transient analysis
RestDesk	Restore window positions
Resume	Resume paused simulation
RotInst	Rotate component or block while placing, copying or pasting
Run	Run simulation
RunSIMPLIS	Runs the SIMPLIS simulator
Save	Save selected schematic

SaveAs	Save selected schematic to specified file. (Interactive if no file specified)
SaveDesk	Save current window positions
SaveGraph	Save a graph to a file
SaveGroup	Save group (simulation data)
SaveRhs	Create nodeset file to speed DC convergence
SaveSnapShot	Saves the current state of a transient analysis to a snapshot file
SaveSymbol	Save symbol in symbol editor
SaveSymlib	Save symbol library
ScriptAbort	Abort currently executing script
ScriptPause	Pause currently executing script
ScriptResume	Resume paused script
ScriptStep	Single step script
Select	Select schematic items (Interactive)
SelectCurve	Select specified curve
SelectGraph	Switches the graph tabbed sheet
SelectLegends	Selects or unselects all graph window legends
SelectSimulator	Selects current simulator for selected schematic
Set	Set option
SetCurveName	Change curve name
SetGraphAnnoProperty	Change a graph object property value
SetGroup	Change current group
SetHighlight	Highlights or unhighlights schematic objects
SetOrigin	Set origin of symbol in symbol editor
SetReadOnly	Sets a vector to be read-only
SetRef	Change/attach reference to variable
SetSnapGrid	Sets schematic snap grid
SetSymbolOriginVisibility	Controls the visibility of the origin marker in the symbol editor
SetToolBarVisibility	Sets the visibility of a toolbar
SetUnits	Change physical units of variable
SetWireColour	Change colour of selected wires
Shell	Execute external application or system command
Show	Display or write to file specified variable
ShowCurve	Show hidden curve

ShowSimulatorWindow	Display simulator status box
SizeGraph	Zoom or scroll graph
TemplateEditProperty	Template script command. Edits the property of a schematic instance
TemplateSetValue	Template script command. Sets the template value
TextWin	Show/hide/toggle schematic text window
Title	Change title of graph or schematic
Trace	Define trace (live graphing during simulation)
Undo	Undo schematic operation
UndoGraphZoom	Restore previous graph view area
UnHighlightCurves	Unhighlights all curves
UnLet	Delete variable
Unprotect	Unprotect and select all protected schematic instances
Unselect	Unselect all schematic instances
UnSet	Delete option
UpdateAllSymbols	Conditionally updates all symbols in open schematics
UpdateSymbol	Updates specified symbol on selected schematic
ViewFile	View file in read-only mode
Wait	Wait for mouse click. (Interactive)
Wire	Start/continue schematic wire. (Interactive unless /loc specified)
WireMode	Enter/exit schematic wiring mode
WriteImportedModels	Write referenced models of netlist to specified file
Zoom	Zoom selected schematic

Commands by Application

Printing

ClosePrinter
 NewPrinterPage
 OpenPrinter
 PrintGraph
 PrintSchematic

Schematic

AddArc	DrawArc	RebuildSymbols
AddCirc	DrawPin	Redo
AddPin	EditCopy	RotInst
AddProp	EditCut	Save
AddSeg	EditPaste	SaveAs
AddSymbolProperty	EditPin	SaveSymbol
Anno	EndSym	SaveSymLib
AppendTextWindow	ImportSymbol	Select
Cancel	Inst	SelectSimulator
ChangeArcAttributes	MakeSymbolScript	SetOrigin
ChangeSymbolProperty	Message	SetSnapGrid
CloseSheet	Move	SetSymbolOriginVisibility
CompareSymbolLibs	MoveProperty	SetWireColour
Copy	Netlist	TextWin
CopyClipSchem	NewSchem	TemplateEditProperty
CopyLocalSymbol	NewSymbol	TemplateSetValue
CreateSym	NoUndo	Undo
Delete	PinDef	Unprotect
DeleteAxis	PrintSchematic	Unselect
DelSymLib	Probe	UpdateAllSymbols
Detach	OpenSchem	UpdateProperties

Schematic

DeleteSymbolProperty	Pan	UpdateSymbol
DelProp	Prop	Wire
DelSym	Protect	WireMode
		Zoom

Graph

AddCurveMarker	DelLegendProp	SaveGraph
AddFreeText	GraphZoomMode	SelectCurve
AddGraphDimension	HideCurve	SelectGraph
AddLegend	HighlightCurve	SelectLegends
AddLegendProp	MoveCurve	SetCurveName
AddTextBox	NewAxis	SetGraphAnnoProperty
CloseGraphSheet	NewGraphWindow	SetHighlight
CopyClipGraph	NewGrid	ShowCurve
Curve	NoPaint	SizeGraph
DelCrv	OpenGraph	Trace
DeleteAxis	PlaceCursor	UndoGraphZoom
DeleteGraphAnno	Plot	UnHighlightCurves

Shell/UI

Arguments	Focus	RedirectMessages
ClearMessageWindow	FocusShell	RegisterUserFunction
CreateToolBar	Hint	RepeatLastMenu
CreateToolButton	HourGlass	ScriptAbort
DefineToolBar	ListOptions	ScriptPause
DefKey	ListStdButtonDefs	ScriptResume
DefMenu	ListStdKeys	ScriptStep
DelMenu	MessageBox	Set
Echo	MoveMenu	SetToolBarVisibility
EditFile	OptionsDialog	Shell
Execute	Redirect	UnSet

Miscellaneous

About	Help	SaveDesk
Close	MoveFile	Title
CreateFont	Quit	ViewFile
EditColour	RestDesk	Wait
EditFont		

Vectors/Groups

CollectGarbage	Let	SetGroup
CreateGroup	MakeAlias	SetReadOnly
DelGroup		SetRef
Discard	OpenGroup	SetUnits
Display	OpenRawFile	Show
KeepGroup	SaveGroup	UnLet

Simulator

CloseSimplisStatusBox	ReadLogicCompatibility	RunSIMPLIS
Listing	Reset	SaveRhs
OpenSimplisStatusBox	RestartTran	SaveSnapshot
Pause	Resume	ShowSimulatorWindow
PreProcessNetlist	Run	WriteImportedModels

File

Cd	MakeTree	MD
CopyFile	MCD	RD
Del		

Lib

ListModels	LoadModelIndex	MakeCatalog
RenameLibs		

Chapter 6 Command Reference

Abort

Abort

Aborts the current simulation. Abort performs the same action as Pause followed by Reset. It stops the current run and then deletes all data associated with it except for any simulation vectors.

Note that this command can only be executed by an assigned key or menu with the direct execution option specified.

AbortSIMPLIS

AbortSIMPLIS

Sends a signal to the SIMPLIS simulator instructing it to abort.

Note that this command can only be executed by an assigned key or menu with the direct execution option specified.

About

About

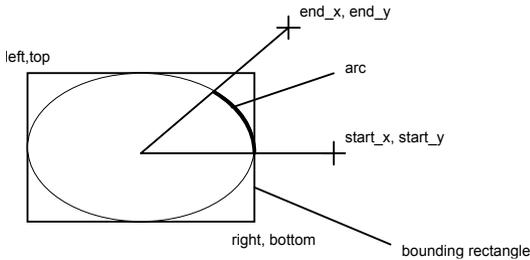
Displays the *about box* which provides version and copyright information.

AddArc

AddArc *left top right bottom start_x start_y end_x end_y*

AddArc is a Symbol Definition Command. It is used to create whole circles and ellipses as well as arcs of circles and ellipses.

The command line arguments are integers describing symbol co-ordinates and all are compulsory. Their meaning is described by the following diagram:



The arc drawn by this command is a segment of an ellipse specified by a bounding rectangle described by the first four arguments. The last four arguments describe two lines drawn from the centre of the ellipse which specify the start and end of the arc. The arc is drawn anti clockwise.

Note that it is better to define a complete 360 degree circle (or ellipse) as two 180 degree arcs. 360 degree circles, where the start and end are coincident or near coincident do not always work reliably with some printer drivers.

See Also

[“Schematic Symbol Script Definition” on page 442](#)

AddCirc

AddCirc *x_org y_org radius*

AddCirc is a Symbol Definition Command. Creates a circle.

<i>x_org</i>	x co-ordinate of circle centre
<i>y_org</i>	y co-ordinate of circle centre
<i>radius</i>	radius of circle.

AddCurveMarker

AddCurveMarker *curve-id division x-position y-position label [length [angle]]*

Adds a curve marker to the currently selected graph sheet. A curve marker is a graph annotation object and its purpose is to label a curve for the purposes of identification or to highlight a feature. See [“Graph Objects” on page 448](#) for more information.

<i>curve-id</i>	Id for curve to which marker will be attached.
<i>division</i>	Division of curve if curve-id refers to a curve group created by a multi-step run. Divisions are numbered from 0 up to 1 minus the number of curves in the group. For single curves set this to zero.
<i>x-position</i>	X-axis location of marker.
<i>y-position</i>	Y-axis location of marker. This is only used if the curve is non monotonic and has more than one point at <i>x-position</i> . The marker will be placed at the point on the curve with the y-axis value that is nearest to <i>y-position</i> .
<i>label</i>	Label for marker. This may use symbolic values enclosed by '%'. See page 450 for details.
<i>length</i>	Length of marker line in <i>view units</i> . See page 460 for an explanation of view units and the view co-ordinate space. If omitted <i>length</i> defaults to 0.1.
<i>angle</i>	Angle of the marker line in the <i>view co-ordinate space</i> (See page 460). Default is 45°

AddFreeText

AddFreeText [/font *font-name*] [/align *align*] *text* [*x-pos* [*y-pos*]]

Adds a free text item to the currently selected graph sheet. Free Text is a graph annotation object. See [page 448](#) for full details.

<i>font-name</i>	Name of font object to be used for text object. This must either be a standard font (as listed in menu File Options Font...) or a font created with the CreateFont command. See page 354 for details.
<i>align</i>	Integer that specifies alignment of text. Possible values: <ul style="list-style-type: none"> 0 Left bottom 1 Centre bottom 2 Right bottom 4 Left base line 5 Centre base line 6 Right base line 8 Left top 9 Centre top 10 Right top 12 Left middle 13 Centre middle 14 Right middle
<i>text</i>	The text to be displayed
<i>x-pos</i>	x-co-ordinate of the text in <i>view units</i> (See page 460). Default = 0.5
<i>y-pos</i>	y-co-ordinate of the text in <i>view units</i> (See page 460). Default = 0.5

AddGraphDimension

AddGraphDimension [/vert] [/label *label*] *curve-id1*
[*pos1* [*curve-id2* [*pos2*]]]

Adds a dimension object to a graph. The dimension object is not yet supported by the GUI.

<i>/vert</i>	If present, a vertical dimension is displayed, otherwise it will be horizontal.
<i>label</i>	Text to add to the dimension object
<i>curve-id1</i>	Id of first curve
<i>pos1</i>	Initial position on curve of dimension. X value if horizontal, otherwise a Y value

<i>curve-id2</i>	Id of second curve
<i>pos2</i>	Initial position on second curve of dimension. X value if horizontal, otherwise a Y value

AddLegend

AddLegend [/autoWidth] [/font *fontName*] [*label* [*x-pos* [*y-pos* [*width* [*height*]]]]]

Adds a legend box to the currently selected graph. A “Legend Box” is a graph annotation object which consist of a rectangle containing a list of curve labels. See “Graph Objects” on page 448 for more information.

<i>/autoWidth</i>	If specified, the width of the box will be adjusted automatically according to its contents.
<i>fontName</i>	Specifies a font to use for the text contained in the box. Must be either a standard font name or one created using the CreateFont command.
<i>label</i>	This is the text that will copied to each entry. To be meaningful this must contain a symbolic value enclosed by '%'. Symbolic values for graph objects are explained more fully on page 450. The default value for <i>label</i> if omitted if %DefaultLabel%. This will result in the curves name and measurements being displayed in the legend box. Some alternatives are: %Curve:Label% displays just the label with no measurements %Curve:Measurements% displays just the measurements %Curve% displays the curve's ID only %Curve:Label%/%Curve:YUnit% displays the curve name and y-axis units
<i>x-pos</i>	X position of box in <i>view units</i> (See page 460). If the value is 1.0 or greater, the box will be placed such that its left hand edge is to the right of the graph's grid area. Default = 0
<i>y-pos</i>	Y position of box in <i>view units</i> (See page 460). If the value is 1.0 or greater, the box will be placed such that its bottom edge is above the graph's grid area. Default = 1
<i>width</i>	Physical width of box in mm. (For CRT monitors this won't be exact. They are typically assumed to be 75 pixels/inch so 1mm is approx. 3 pixels). Note that this value will be ignored if <i>/autowidth</i> is specified. Default = 50.

height Physical height of box in mm. (See notes above wrt CRT monitors)

AddLegendProp

AddLegendProp *curve_id property_name property_value*

Adds a property to a graph legend. Legend properties are generally used to display measurement information for a curve. Their name and value is displayed below a curve's legend (or label).

curve_id Curve Id. Curve id is returned by the functions GetSelectedCurves (see [page 187](#)), GetAxisCurves (see [page 146](#)) and GetAllCurves (see [page 143](#))

property_name Name of property. May be any string and may contain spaces.

property_value Value of property. May be any string and may contain spaces.

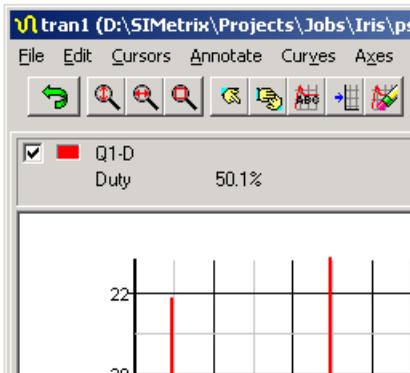
Example

The following is extracted from the script `curve_duty` which displays duty cycle for selected curves.

```
let curves=GetSelectedCurves()  
let numCurves = length(curves)  
  
...  
  
for idx=0 to numCurves-1
```

Script lines to retrieve duty cycle ...

```
AddLegendProp {curves[idx]} "Duty cycle" {duty_cycle}  
  
next idx
```



A typical result is displayed above. In this example the property name is 'Duty' and its value is 50.1%

AddPin

AddPin *pin_name pin_number x y [label_x label_y label_flags]*
 [*qualifier_list*]

AddPin is a Symbol Definition Command. A pin is a point on a symbol where wires can be connected. Refer to “Schematic Symbol Script Definition” on page 442 for more details.

<i>pin_name</i>	Text string. Any pin name can be used as long as it does not contain spaces. However, in order to allow the plotting of currents from the schematic, certain pin names must be used for primitive components.
<i>pin_number</i>	Integer. Determines the order in which the pins appear on the device's netlist entry. Must be in a certain order for primitive components.
<i>x, y</i>	Integer. Symbol co-ordinates of pin. As wires always snap to grid points pins must lie on grid points if it is to be possible to connect to them. This means that the x and y co-ordinates must be a multiple of 100.
<i>label_x, label_y</i>	X and Y position relative to pin of pin label. Text of label will be pin name. Scaling is 100 points per grid square. Justification is determined by <i>label_flags</i> - see below.
<i>label_flags</i>	Justification of pin label text. Values as follows: <ul style="list-style-type: none"> 0: left top 1: centre top 2: right top 8: left baseline 9: centre baseline 10: right baseline Baseline means the base for upper case characters. The tails of some lower case characters go below the baseline.
<i>qualifier_list</i>	One or more qualifiers used for XSPICE devices. For more information refer to the “Using XSPICE Devices” section in the “Simulator Devices” chapter of the <i>Simulator Reference</i>

Manual. Qualifiers may be one of

vecclose	Pin closes a vector connection. This causes a ']' to be placed after the pin's connection in the netlist
vecopen	Pin opens a vector connection. This causes a '[' to be placed before the pin's connection in the netlist
vecopenl	As vecopen except that it forces the '[' to always be placed before any other qualifiers.
invert	Inverts a digital pin. Places a '~' before it in the netlist
%d	Forces pin to be of digital type.
%g	Forces pin to be of type "grounded conductance".
%gd	Forces pin and the one following to be of type "differential conductance"
%h	Forces pin to be of type "grounded resistance".
%hd	Forces pin and the one following to be of type "differential resistance"
%i	Forces pin to be of type "single ended current".
%id	Forces pin and the one following to be of type "differential current"
%v	Forces pin to be of type "single ended voltage".
%vd	Forces pin and the one following to be of type "differential voltage"

See Also

["Schematic Symbol Script Definition" on page 442](#)

AddProp

AddProp /font *fontIndex* /sel *name* [*initvalue*] [*flags*] [*X_pos* *Y_pos*]

AddProp is a Symbol Definition Command. A Property is a text string that can be attached to a symbol which is normally used to describe a special characteristic such as a component reference or value. A comprehensive explanation on properties can be found in the "Schematic Editor" chapter of the User's manual.

name Text string. This can be anything but usually would be one of the special properties which convey a special meaning. A full listing of these is given in the "Schematic Editor" chapter of the

User's manual. The important ones are listed below

ref	Component reference.
value	Component value or model name. (E.g. BC547)
model	Single letter to signify type of device.
netname	If present forces netlister to assign value of value property to all nets connected to component. This property is used by the 'Terminal' component in the Symbols menu.
schematic_path	File system pathname for a hierarchical block
valuescript	Script that is called when F7 is pressed or the menu Edit Value/Model is selected.

Some other property names are used in scripts such as *biasv* which is used by the bias point annotation scripts and is attached to the bias point annotation markers.

initvalue Text string, integer or real. The initial value of the property when the component is first placed. It may be changed subsequently with the Prop command. Examples: the value of a ref property would be something like 'R23' or 'Q4'. The value of a value property maybe '33k' or 'IRF640'.

flags This is the property's *attribute flags*. It is a single integer that describes a number of attributes for the property. For full information see “Attribute flags” on page 404.

X_pos, Y_pos If specified, the property will be placed at an absolute location specified by X_pos and Y_pos relative to the reference point of the symbol. The flags value specifies the justification of the text as described on page 404

If X_pos and Y_pos are specified, the text will be displayed vertically in 90 and 270 degree rotated orientations.

fontIndex Integer from 1 - 8 that specifies one of 8 fonts as follows:

- 1 Default
- 2 Caption
- 3 Free text
- 4 Annotation
- 5 User 1
- 6 User 2
- 7 User 3
- 8 User 4

The value specified by */font fontIndex* overrides bits 11-13 of the *flags* value (see above).

/sel If specified the property is marked “selectable”. This means that the selection boundary of the instance which owns the property will be extended to include the property text. This is usually used for symbols that consist only of properties and have no body.

Examples

```
AddProp ref Q? 26
```

A symbol containing this line in its definition will possess the property of *name* *ref* and when first placed on a schematic will have the initial value of *Q?*. The text *Q?* will be displayed on the schematic to the right of the symbol when in normal orientation and underneath the symbol when in a 90° rotated orientation.

```
AddProp ref Q? 8 100 200
```

The same property as the above example but instead it will be placed 100 units horizontally and 200 unit vertically from the symbol origin. The text of the property will be left justified and positioned vertically referenced to its base line.

See Also

[“Schematic Symbol Script Definition” on page 442](#)

AddSeg

```
AddSeg start_x start_y end_x end_y
```

AddSeg is a Symbol Definition Command. It is used to add a line segment to a symbol.

start_x Integer. Symbol x co-ordinate of start of segment

start_y Integer. Symbol y co-ordinate of start of segment

end_x Integer. Symbol x co-ordinate of end of segment

end_y Integer. Symbol y co-ordinate of end of segment

See Also

[“Schematic Symbol Script Definition” on page 442](#)

AddSymbolProperty

```
AddSymbolProperty name flags value [x y]
```

Adds a property to the symbol currently open in the symbol editor. See the User's Manual for detailed information on properties.

name Property name

<i>flags</i>	Property attribute flags. See “Attribute flags” on page 404
<i>value</i>	Property value
<i>x, y</i>	If both specified the property will automatically be given a fixed position attribute and will be located at the position given. The position is relative to the symbol's origin

AddTextBox

AddTextBox [/font *font-name*] *text* [*x-position* [*y-position*]]

Adds a Text Box to the currently selected graph. A text box is an item of text enclosed by a border.

<i>font-name</i>	Name of font to be used for text. This must either be a built in font or one created using CreateFont.
<i>text</i>	Text to be displayed in the box. This may use symbolic value enclosed by '%'. The following are meaningful for Text Box objects: <ul style="list-style-type: none"> %Date% The date when the object was created %Time% The time when the object was created %Version% The name and current version of the program <p>See page 450 for more information on symbolic values</p>
<i>x-position</i>	The x position of the box in <i>view units</i> (See page 460)
<i>y-position</i>	The y position of the box in <i>view units</i> (See page 460)

Anno

Anno [/prop *property_name*] [/nopaint] [/bypos] [/minSuffix *min_suffix*]

Automatically allocates unique component references to all components on currently selected schematic.

/prop <i>property_name</i>	If specified, annotates properties of name <i>property_name</i> . Otherwise properties of name “ref” are annotated.
/nopaint	The anno command always forces the schematic window to refresh if any changes to properties were made. This action is inhibited if this switch is specified. This is usually used if the property being annotated is hidden and therefore will cause no visual change.
/minSuffix <i>min_suffix</i>	Minimum suffix value that will be used for new references. The anno command works by locating duplicate references then searching for the first suffix value that resolves the duplicate. The minSuffix switch specifies the lowest value that will be used. So if set to 100 for example, the lowest resistor reference

would be R100. Note that this will not force existing references to be updated to values greater than *min_suffix*. Only values that need changing will be affected.

/bypos If specified, all references will be reassigned according to their position on the schematic working left to right. Unlike */minSuffix* this switch does reassign all references. It can be used with */minSuffix* to reassign all references in a schematic according to a desired specification.

Typically Anno is used prior to running the Netlist command. The latter requires unique references to function.

Note that Anno will not allocate a new reference to a component unless it is necessary to do so to avoid a duplication. When there is a duplication, the component which was most recently added to the schematic will be modified.

AppendTextWindow

AppendTextWindow [*/file filename*] [*/copy*] [*text*]

Inserts text into the schematic editor's *simulator command window* also known as the *F11 window*

filename If specified, the contents of the specified file is placed in the F11 window

/copy If specified the *text* is copied to the F11 window replacing the existing text. Otherwise *text* is appended.

text If '*filename*' is absent, *text* is inserted in the F11 window

Notes

Text is always is always appended to the end of the window's existing contents.

See Also

["WriteF11Lines" on page 321](#)

Arguments

Arguments [*argument_list*]

Declares arguments for a script. Full details for passing arguments to scripts are given in ["Script Arguments" on page 45](#).

argument_list List of arguments to be used in the script in the order in which they are passed. Arguments that are *passed by reference* should be prefixed with '@'.

BuildDefaultOptions

BuildDefaultOptions

Resets preference settings to factory defaults

Cancel

Cancel

Cancel current schematic editing operation (wiring, moving etc.). As the command line is inactive while editing operations are in progress this command is only of value when used in a key or menu definition with the flag set to 5 or with /immediate switch for DefMenu command. For more information see “[User Defined Key and Menu Definitions](#)” on page 434.

Cd

Cd [*directory_name*]

Cd is almost identical to the DOS cd or chdir commands. It changes the current directory to that specified. Unlike the DOS command, however, it will also change the current drive if it is included in the directory name. If no directory name is specified, the current directory will be displayed.

ChangeArcAttributes

ChangeArcAttributes [*theta*] [*v_over_h*]

Modifies the attributes of the selected arc or arcs in the currently open symbol editor sheet.

theta Arc swept angle in degrees. Default = 90.

v_over_h vertical radius/horizontal radius. Default = 1 (i.e. a circular arc)

ChangeSymbolProperty

ChangeSymbolProperty [/value *value*] [/flags *flags*] [/loc *loc*]
[*name*]

Modifies a named or selected symbol editor property. In the symbol editor, pin names are also represented as properties, so this command is also used to edit pin names.

value New property value

flags New property attribute flags. See “[Attribute flags](#)” on page 404

loc New absolute location. If the location was previously relative, this will be changed to absolute if this value is specified.

name If specified the property of the specified name will be modified. Otherwise all selected properties will be modified.

See also

Prop page [page 402](#)

AddProp page [page 344](#)

ClearMessageWindow

ClearMessageWindow

Clears the command shell message window

Close

Close schem

OR

Close graph

The first form closes the selected schematic window, the second form closes all graph windows.

CloseGraphSheet

CloseGraphSheet

Closes the current tabbed sheet in the selected graph window. If the window has only one sheet, the whole window will be closed.

ClosePrinter

ClosePrinter [/abort]

ClosePrinter is one of a number of commands and functions used for non-interactive printing. This is explained in [“Non-interactive and Customised Printing” on page 463](#). Printing sessions are started with OpenPrinter after which print output commands such as PrintGraph and PrintSchematic may be called. The session is terminated with ClosePrinter which actually initiates the printing activity. If the /abort switch is specified, the print job is terminated and no print output will be produced.

See Also

[“NewPrinterPage” on page 392](#)

[“OpenPrinter” on page 395](#)

[“PrintGraph” on page 401](#)

[“PrintSchematic” on page 401](#)

[“GenPrintDialog” on page 141](#)

[“GetPrinterInfo” on page 185](#)

CloseSheet

CloseSheet [/force]

Closes the currently selected schematic or symbol editor tabbed sheet. If the sheet is the last in its window, the window will also be closed.

If /force is specified, the sheet will be closed unconditionally. Otherwise user interaction will be required if the schematic or symbol has not been saved.

CloseSimplisStatusBox

CloseSimplisStatusBox

Closes the SIMPLIS simulation status box.

See Also

“OpenSimplisStatusBox” on page 396

CollectGarbage

CollectGarbage

Deletes temporary vectors. This command is only needed for scripts running endless or very long loops. SIMetrix creates temporary vectors when calculating vector expressions. These do not get deleted until control is returned to the command line. In the case of a script that calculates many expressions, it is possible for the memory used by the temporary vectors to become excessive. Calling CollectGarbage at regular intervals will resolve this problem.

CompareSymbolLibs

CompareSymbolLibs [/detail] *lib1 lib2*

/detail If specified, a detailed report is given when two symbols do not match. Detail about what doesn't match will be provided. This could be mismatched segments, properties or pins.

lib1 Path of first symbol library file

lib2 Path of second symbol library file

Compares two symbol libraries by comparing each symbol in turn. A message will be output for each symbol that is different or is not found in one of the libraries. Symbols are classed as identical if:

1. All graphical elements are identical. Graphical elements are segments and arc segments. (Circles are classed as arc segments)
2. All pins have the same name, location and order
3. All *protected* properties are identical.

Unprotected properties are *not* compared.

If no differences are found the command will output the message “The symbol files are identical”.

Copy

Copy

Initiates the schematic 'copy' editing operation. This performs exactly the same function as the “Duplicate” button on the schematic sheet and the equivalent menu. Note that the clipboard is bypassed for this operation.

CopyClipGraph

CopyClipGraph [/mark] [/mono] [/file *filename*] [/format *format*]
[/vp *viewport_x viewport_y*]

Copies a graphical picture of the graph to the clipboard or to a specified file.

/mark	If specified, markers are displayed on each curve as a means of identification. This is enabled automatically if /mono is specified.
/mono	Copy graph in monochromatic form.
/file <i>filename</i>	If specified, the graph is written output in the format specified by the format switch. If not specified the graph picture is written to the clipboard.
/format <i>format</i>	Picture format used. Choices are: wmf - Enhanced metafile format. (Windows only) svg - “Scalable Vector Graphics” format. A scalable format compatible across platforms. Not supported in clipboard mode jpeg - JPEG format png - PNG format bmp - Windows bitmap format In clipboard mode jpeg, png and bmp do the same thing - that is write an uncompressed bitmapped image of the graph. If /format is omitted, <i>wmf</i> will be used in Windows. In Linux, <i>svg</i> will be used for file output and a bitmapped format will be used for clipboard output.
/vp <i>x y</i>	Viewport dimensions in pixels. This is used to specify the size of the image if a bitmapped format (png, jpeg, bmp) is specified. <i>x</i> is the width, <i>y</i> is the height

Notes

This command makes it possible to export graphs into other windows applications such as word processors. The *clipboard* is a central store within operating system which is accessible by all applications. Refer to system documentation for more information.

CopyClipSchem

CopyClipSchem [/mono] [/file *filename*] [/format *format*] [/vp
viewport_x viewport_y]

Copies a graphical picture of the currently displayed schematic to the clipboard or to a specified file.

<code>/mono</code>	Copy graph in monochromatic form.
<code>/file filename</code>	If specified, the schematic is written output in the format specified by the format switch. If not specified the schematic picture is written to the clipboard.
<code>/format format</code>	Picture format used. Choices are: wmf - Enhanced metafile format. (Windows only) svg - “Scalable Vector Graphics” format. A scalable format compatible across platforms. Not supported in clipboard mode jpeg - JPEG format png - PNG format bmp - Windows bitmap format In clipboard mode jpeg, png and bmp do the same thing - that is write an uncompressed bitmapped image of the schematic. If /format is omitted, <i>wmf</i> will be used in Windows. In Linux, <i>svg</i> will be used for file output and a bitmapped format will be used for clipboard output.
<code>/vp x y</code>	Viewport dimensions in pixels. This is used to specify the size of the image if a bitmapped format (png, jpeg, bmp) is specified. <i>x</i> is the width, <i>y</i> is the height

Notes

This command makes it possible to export schematics into other windows applications such as word processors. The *clipboard* is a central store within operating system which is accessible by all applications. Refer to system documentation for more information.

CopyFile

CopyFile [*/force*] *from_file to_file*

Copies a file.

<code>/force</code>	If specified, <i>to_file</i> will be overwritten if it exists. Otherwise if <i>to_file</i> exists, the command will fail.
<code>from_file</code>	Source file
<code>to_file</code>	Destination file

CopyLocalSymbol

CopyLocalSymbol *symbol_name* [*new_symbol_name*]

Copies a symbol in the currently selected schematic to the global library.

<i>symbol_name</i>	Name of local symbol to copy
<i>new_symbol_name</i>	New name for symbol when copied to global library. If omitted, the original name is used. If the symbol already exists in the global library, an error will be raised.

CreateFont

CreateFont *font-name font-base*

Creates a new font object based on an existing font. The name given to the font can be used to specify the font for some graph annotation objects. Once CreateFont is called, its name will be displayed in the list displayed when the File|Options|Font... menu is selected.

<i>font-name</i>	Name of new font
<i>font-base</i>	Name of font to be used to set initial properties. May be any font listed in the menu File Options Font... or one of the following: Standard, StandardMedium or StandardLarge.

CreateGroup

CreateGroup [/title *title*] *label*

Creates a data group. All vectors (or variables) are organised into groups. Each simulation run creates a new group and all data for that simulation is placed there. For more information, see [“Groups” on page 37](#).

<i>label</i>	Base name of group. The actual group name will be appended by a number to make it unique. The new group will become the <i>current</i> group. To find the name actually used, you can call the function Groups (page 210) immediately after calling this command. The first element of Groups (i.e. (Groups())[0]) is always the current group.
<i>title</i>	Optional title. This will be displayed in the box displayed when selecting a Change Data Group... menu. It is also returned by a call to Groups('title')

See Also

[“DelGroup” on page 373](#)

CreateSym

CreateSym [/local | /file *libfile*] [/flags *flags*] *symbol_name*
[*symbol_description* [*catalogue*]]

CreateSym is a Symbol Definition Command. All symbol definitions must start with this command and finish with EndSym.

<i>symbol_name</i>	Text string. Name of symbol being defined. This can be anything not already used in a previous symbol definition and must not contain spaces. This is known as the “internal name” in the user interface.
<i>symbol_description</i>	Text string. Description of symbol. If specified this will appear in the choose symbol dialog box opened by the menu Place From Symbol Library... (This menu calls the GetSymbols function). This is known as the “user name” in the user interface.
catalogue	This permits the implementation of multiple catalogues for symbols. This is a method of categorising symbols so that they can be easily located. The menu Place From Symbol Library... lists available symbols in a tree structure and the catalogue name is used to define its location in that tree. Branch names are separated by semi-colons. E.g. “Digital;Flip-flops” creates a top level called “Digital” and a sub-branch called “Flip-flops”.
/local	If specified, the symbol will be created in the currently open schematic and will not be saved to the global library.
/file libfile	If specified, the symbol will be saved to libfile. If neither /file nor /local are specified, the symbol will be saved to the file default.xslb in the SymbolLibs directory.
/flags flags	If flags=1 then the symbol will be stored with tracking enabled. This means that any existing instances of the symbol with the specified name will be automatically be updated when the symbol is edited.

CreateToolBar

CreateToolBar *window_name toolbar_name* [*caption* [*visibility*]]

Creates a new empty toolbar. To add buttons to the toolbar use command “DefineToolBar” on page 362.

<i>window_name</i>	Name of window where toolbar is to reside. Must be one of:	
	CommandShell	Command shell window
	Schematic	Schematic windows
	Symbol	Symbol editor windows
	Graph	Graph windows
<i>toolbar_name</i>	User assigned name for toolbar. You can use any name that doesn't clash with a pre-defined toolbar name as defined in the table below. The name must not contain spaces.	

Pre-defined toolbars:	
CommandShellMain	Command Shell toolbar
SchematicMain	Main schematic toolbar
SchematicFile	Schematic file operations toolbar
SchematicComponents	Schematic component toolbar (simetrix mode)
SIMPLISComponents	Schematic components toolbar (SIMPLIS mode)
SymbolMain	Symbol editor toolbar
GraphMain	Graph window toolbar

This name is used to reference the tool bar in the DefineToolBar and SetToolBarVisibility commands.

caption Optional caption for toolbar. This is displayed in the caption bar of the toolbar that is visible when the toolbar is 'undocked'.

visibility Specifies when the tool bar is visible. This can be subsequently changed using the SetToolBarVisibility command. Possible values are:

always	toolbar is always visible
never	toolbar is never visible
simplis	(schematics only) toolbar is only visible in SIMPLIS mode
simetrix	(schematics only) toolbar is only visible in SIMetrix mode

See Also

- ["CreateToolButton" on page 356](#)
- ["DefButton" on page 361](#)
- ["SetToolBarVisibility" on page 423](#)
- ["GetToolButtons" on page 204](#)

CreateToolButton

CreateToolButton [/toggle] [/class *class_name*] [/shortcut *key*] *name*
graphic [*hint*]

Creates or redefines a tool bar button. This command creates the properties of the button but not the command it executes when it is pressed. To define the command, use [DefButton \(page 361\)](#),

/toggle If specified, the button will have a toggle action and will have two commands associated with it. One command will be

	executed when the button is pressed and another when it is released. The 'Wire' pre-defined button is defined in this manner
<i>class_name</i>	This is used with the function GetToolButtons (page 204) to select buttons according to their function. Set this value to 'component' if you wish the button to be displayed in the GUI which selects component button.
<i>key</i>	Specifies a shortcut key that will perform the same action as the tool button. For key codes see " DefKey " on page 365
<i>name</i>	Name of button. This may be one of the pre-defined types described in " DefineToolBar " on page 362 in which case this command will redefine its properties. You may also specify a new name to create a completely new button.
<i>graphic</i>	Graphical image to be displayed on the button. This may be one of the pre-defined images listed in " DefineToolBar " on page 362 or you may use a user defined image specified in a file. The file must be located at the following location: Windows: <i>simetrix-root</i> \support\images Linux: <i>simetrix-root</i> /share/images where <i>simetrix-root</i> is the top level directory in the SIMetrix tree. The file may use windows bitmap (.bmp), portable network graphic (.png) or JPEG (.jpg) formats. The PNG format supports masks and this format must be used if transparent areas are needed in the graphic. If no mask is found in the graphic file, one will be created which will make the area outside the outer perimeter transparent.
<i>hint</i>	Text that describes the operation of the button. This will be displayed when the user passes the mouse cursor over the button.

See Also

- ["CreateToolBar" on page 355](#)
- ["SetToolBarVisibility" on page 423](#)
- ["GetToolButtons" on page 204](#)

CursorMode

CursorMode on | off |toggle | step | stepref | stepShift | stepRefShift

Switches cursor mode of selected graph. In *cursor mode*, two cursors are displayed allowing measurements to be made. See the User's manual for more information on cursors.

on	Switch cursors on
off	Switch cursors off

toggle	Toggles on off
step	Step cursor to next curve
stepref	Step reference cursor to next curve
stepShift	Steps cursor to next curve within a group. Curves are grouped - for example - for Monte Carlo runs.
stepRefShift	Steps reference cursor to next curve within a group. Curves are grouped - for example - for Monte Carlo runs.

Curve

```
Curve  [/autoXlog]
      [/autoYlog]
      [/autoAxis]
      [/axisId axis_id]
      [/bus type]
      [/coll]
      [/dig]
      [/icb clipboard_index]
      [/loglog]
      [/name curve_name]
      [/newAxis]
      [/newGrid]
      [/newSheet]
      [/new]
      [/select]
      [/title title]
      [/xauto]
      [/xdelta x_grid_spacing]
      [/xl x_low_limit x_high_limit]
      [/xlabel x_label_name]
      [/xlog]
      [/xunit x_unit_name]
      [/yauto]
      [/ydelta y_grid_spacing]
      [/yl y_low_limit y_high_limit]
      [/ylabel y_label_name]
      [/ylog]
      [/yunit y_unit_name]

      [y_expression]
      [x_expression]
```

Curve can be used to add a new curve to an existing graph created with Plot or to change the way it is displayed.

/autoXlog	Only effective when graph sheet is empty. If specified, the x-axis will be logarithmic if the x-values are logarithmically spaced.
/autoYlog	Only effective when graph sheet is empty. Same as /autoXlog except that if x-values are logarithmically spaced, the Y axis will be logarithmic
/autoAxis	<p>If specified, the new curve will be added to a compatible axis according to its physical units i.e Voltage, Current etc. The rules used are as follows:</p> <p>If the currently selected axis or grid (shown by black axis line) has the same units as curve to be plotted or if it has undefined units (designated by a '?' on label), that axis will be used.</p> <p>If any other axis or grid has compatible units (i.e same as curve or undefined) that axis will be used.</p> <p>If no axes exist with compatible units, a new axis (not grid) will be created to accommodate the curve.</p>
/axisId <i>axis_id</i>	If specified, the new curve will be added to a y-axis with the id specified by <i>axis_id</i> . Axis id is returned by the functions GetAllYAxes (page 144), GetCurveAxis (page 152) and GetSelectedYAxis (page 188). These are documented in the "Script Reference Manual". This is available as a PDF file on the install CD. A hardcopy version is also available for an additional charge.
/bus <i>type</i>	If specified, the new curve will be plotted on a digital axis and will be plotted as a <i>bus</i> curve. <i>type</i> may be 'hex', 'dec' or 'bin' specifying hexadecimal, decimal or binary display respectively.
/coll	Does nothing. For compatibility with version 3.1 and earlier.
/dig	If specified, new curve will be plotted on new digital axis. Digital axes are stacked on top of main axes and are sized and labelled appropriately for digital waveforms.
/icb <i>clipboard_index</i>	Specifies the internal clipboard as the source of the curve data. <i>clipboard_index</i> is a value of 0 or more that indicates which curve in the internal clipboard is to be used. The function HaveInternalClipboardData (page 212) may be used to determine the number of curves available. The maximum acceptable value for <i>clipboard_index</i> is thus one less than the value returned by HaveInternalClipboardData.
/loglog	Only effective when graph sheet is empty. Forces both y and x axes to be
/name <i>curve_name</i>	If specified, curve will be named <i>curve_name</i> .
/newAxis	If specified, the new curve will be plotted on a new y-axis.
/newGrid	If specified, the new curve will be plotted on a new grid.

Script Reference Manual

<i>/newSheet</i>	Does nothing. Included for compatibility with Plot command.
<i>/newWindow</i>	Does nothing. Included for compatibility with Plot command.
<i>/select</i>	If specified, the new curve will be selected.
<i>/title title</i>	Does nothing. Included for compatibility with Plot command.
<i>/xauto</i>	Flag. Use automatic limits for x-axis. If this appears after a /xl specification /xauto will override it and vice-versa.
<i>/xdelta</i>	Specify spacing between major grid lines on x-axis. Followed by...
<i>x_grid_spacing</i>	Real. For default spacing use '0'.
<i>/xl</i>	Use fixed limit for x-axis. Followed by ...
<i>x_low_limit</i>	Real. Lower limit of x-axis.
<i>x_high_limit</i>	Real. Higher limit of x-axis.
<i>/xlabel</i>	Specify label for x-axis. Followed by...
<i>x_label_name</i>	Text String. Label name. If it contains spaces, whole string must be enclosed in quotes ("").
<i>/xlog</i>	Only effective when graph sheet is empty. Forces logarithmic x-axis.
<i>/xunit</i>	Specify units for x-axis (Volts, Watts etc.). Followed by ...
<i>x_unit_name</i>	Text string. Unit name. If it contains spaces, the whole string must be enclosed in quotes (""). You should not include an engineering prefix (m, K etc.).
<i>/yauto</i>	Flag. Use automatic limits for y-axis. If this appears after a /yl specification /yauto will override it and vice-versa.
<i>/ydelta</i>	Specify spacing between major grid lines on y-axis. Followed by...
<i>y_grid_spacing</i>	Real. For default spacing use '0'.
<i>/yl</i>	Use fixed limit for y-axis. Followed by ...
<i>y_low_limit</i>	Real. Lower limit of y-axis.
<i>y_high_limit</i>	Real. Higher limit of y-axis.
<i>/ylabel</i>	Specify label for y-axis. Followed by...
<i>y_label_name</i>	Text String. Label name. If it contains spaces, whole string must be enclosed in quotes ("").
<i>/yunit</i>	Specify units for y-axis. Followed by ...
<i>y_unit_name</i>	Text string. Unit name. Other comments as for x unit name.
<i>y_expression</i>	Text string. Expression describing curve to be added to graph.
<i>x_expression</i>	Text string. Expression describing x values of curve defined by y expression. If omitted, reference of <i>y_expression</i> will be used.

CurveEditCopy

CurveEditCopy *curve-id1* [*curve-id2 ...*]

Copy specified curves to the internal clipboard. Curves so copied may be subsequently plotted using the command [Curve](#) (page 358) with the /icb switch.

curve-idn Id of curve. A number of functions return this value including [GetSelectedCurves](#) (page 187).

See Also

[Curve](#) (page 358)
[HaveInternalClipboardData](#) (page 212)

DefButton

DefButton [/immediate] [/comgroup *command_group*] *button_name*
command [*up_command*]

Defines the command executed when a button is pressed.

/immediate If specified, the command will be enabled for immediate execution. That is the command will be executed immediately even if another command - such as a simulation run - is currently in progress. This will only be accepted when the command specified is one of a small number of built-in command enabled for immediate execution. For the list of commands, see “[DefMenu](#)” on page 368. You may not call a script if immediate execution is specified.

command_group This can be used with the function [GetLastCommand](#) (page 172). [GetLastCommand](#) returns the text of the most recent command executed which specified the supplied command group value. The command [DefMenu](#) (page 368) also uses this feature.

button_name Name of button. Either a pre-defined button as listed in “[DefineToolBar](#)” on page 362 or a new button created with [CreateToolButton](#) (page 356).

command Command to be executed when the button is pressed. If */immediate* is *not* specified this may be any valid command including a script.

up_command Command to be executed when a toggle button is released. The button must be defined to have a toggle action using the */toggle* switch for the [CreateToolButton](#) (page 356) command.

See Also

“[SetToolBarVisibility](#)” on page 423
 “[GetToolButtons](#)” on page 204

DefineToolBar

DefineToolBar *toolbar_name* *button_defs*

Defines the buttons for a user defined toolbar created using [CreateToolBar \(page 355\)](#). To define the buttons for a pre-defined toolbar, the associated option setting must be set using the command [Set \(page 420\)](#).

toolbar_name Name of toolbar. This must be a toolbar created using [CreateToolBar \(page 355\)](#).

button_defs Semi-colon delimited list of button names to add to the toolbar. Buttons may either be one defined using [CreateToolButton \(page 356\)](#) or one of the pre-defined types shown in the table below. The '-' character may also be used to specify a spacer

Pre-defined buttons:

Button Name	Graphic	Function
AddCurve	newcurve.bmp	Add Curve
AddFourier	newfourier.bmp	Fourier...
CalcAveragePower	avg.bmp	Display Average Power/Cycle
CalcFall	falltime.bmp	Display Fall Time
CalcHighPass3db	3dbhighpass.bmp	Display -3dB Point (High Pass)
CalcLowPass3db	3dblowpass.bmp	Display -3dB Point (Low Pass)
CalcRise	risetime.bmp	Display Rise Time
CalcRMS	rms.bmp	Display RMS/Cycle
Capacitor	cap.bmp	Place Capacitor
Copy	copy.bmp	Duplicate
Delete	erase.bmp	Cut
DeleteAxis	delgrid.bmp	Delete Axis/Grid
DeleteCurve	delete.bmp	Delete Curve
Diode	diode.bmp	Place Diode
Flip	flip.bmp	Flip
GraphClose	fileclose.bmp	Close Graph
GraphOpen	fileopen.bmp	Open Graph
GraphSave	filesave.bmp	Save Graph

Button Name	Graphic	Function
Ground	gnd.bmp	Place Ground
HideCurves	hide.bmp	Hide Selected Curves
IGBT	igbt.bmp	Place IGBT
Inductor	ind.bmp	Place Inductor
IProbe	iprobe.bmp	Place Current Probe
ISource	isource.bmp	Place Current Source
Mirror	mirror.bmp	Mirror
MoveCurve	movecurve.bmp	Move Curve to Selected Axis/Grid
NewAxis	newaxis.bmp	New Axis
NewGrid	newgrid.bmp	New Grid
NJFET	njfet.bmp	Place N-channel JFET
NMOS	nmos.bmp	Place N-channel MOSFET
NMOS3IC	nmos_ic3.bmp	Place 3 term N-channel MOSFET
NMOS4	nmos_ic.bmp	Place 4 term N-channel MOSFET
NPN	npn.bmp	Place NPN Transistor
Opamp	opamp.bmp	Place Opamp
Options	options.bmp	Options
PJFET	pfjet.bmp	Place P-channel JFET
PMOS	pmos.bmp	Place P-channel MOSFET
PMOS3IC	pmos_ic3.bmp	Place 3 term P-channel MOSFET
PMOS4	pmos_ic.bmp	Place 4 term P-channel MOSFET

Button Name	Graphic	Function
PNP	pnp.bmp	Place PNP Transistor
Print	print.bmp	Print
PSU	psu.bmp	Place PSU
Resistor	res.bmp	Place Resistor (Box shape)
ResistorZ	resz.bmp	Place Resistor (Z shape)
Rotate	rotate.bmp	Rotate
SatInd	sat_ind.bmp	Place Saturable Inductor
SatTx	tx_sat.bmp	Place Saturable Transformer
SchemClose	fileclose.bmp	Close Schematic
SchemNew	newschem.bmp	New Schematic
SchemOpen	fileopen.bmp	Open Schematic
SchemSave	filesave.bmp	Save Schematic
SchemSaveAll	saveall.bmp	Save All Schematics
SCR	scr.bmp	Place Thyristor
ShowCurves	show.bmp	Show Selected Curves
SimPause	pause.bmp	Pause Simulation
SimRunNetlist	run.bmp	Run Netlist
SimRunSchem	run.bmp	Run Schematic
SymbolNew	newsymbol.bmp	New Symbol
TitleCurve	curvetitle.bmp	Change Curve Name
TL	tl.bmp	Place Transmission Line
Tx	tx.bmp	Place Transformer
Undo	undo.bmp	Undo
UndoZoom	undo.bmp	Undo Zoom

Button Name	Graphic	Function
VProbe	vprobe.bmp	Place Voltage Probe
VSource	vsource.bmp	Place Voltage Source
Waveform	vsig.bmp	Place Waveform Generator
Wire	pencil.bmp	Wire Mode
Zener	zener.bmp	Place Zener Diode
ZoomFull	zoomfull.bmp	Fit Window
ZoomIn	zoomin.bmp	Zoom In
ZoomOut	zoomout.bmp	Zoom Out
ZoomRect	zoomrect.bmp	Zoom Box
ZoomXAuto	zoomwidth.bmp	Fit Width
ZoomYAuto	zoomheight.bmp	Fit Height

The graphic images for all pre-defined buttons are built-in to the program, but the image files from which they were created can be found on the install CD.

See Also

[“DefButton” on page 361](#)

[“SetToolBarVisibility” on page 423](#)

[“GefToolButtons” on page 204](#)

DefKey

DefKey *Key_Label* *Command_string* [*option_flag*]

DefKey is used to define custom key strokes.

Key_Label Code to signify key to define. See table below for list of possible labels. All labels may be suffixed with one of the

following:

:SCHEM	Key defined only when a schematic window is currently active
:GRAPH	Key defined only when a graph window is currently active
:SHELL	Key defined only when the command shell is currently active.
:SYMBOL	Key defined only when a symbol editor window is currently active

If no suffix is provided the key definition will be active in all windows.

Command_string

A command line command or commands to be executed when the specified key is pressed. Multiple commands must be separated by semi-colons (;). Unless the command string has no spaces, it must wholly enclosed in double quotation marks (").

option_flag

A number between 0 and 5 to specify the manner in which the command is executed. These are as follows:

- 0, 4 Default. Command is echoed and executed. Any text already in command line is overwritten.
- 5 Immediate mode. Command is executed immediately even if another operation - such as a simulation run or schematic editing operation - is currently in progress. For other options the command is not executed until the current operation is completed. Only a few commands can be assigned with this option. These are

Cancel
DefMenu
DefKey
Echo
Let
Move
Pan
Pause
Quit
RotInst
Select
ScriptAbort
ScriptPause
ScriptResume
Shell
Wire
Zoom

Note, the Let command can be used to set a global variable which can then be tested in running script. This is a

convenient method of providing user control of script execution.

Valid key labels:

Function keys:

F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12

INS Insert key
DEL Delete key
HOME Home key
END End key
PGUP Page up key
PGDN Page down key
LEFT ←
RIGHT →
UP ↑
DOWN ↓
TAB Tab key
BACK Back space

ESC Escape key

NUM1 Keypad 1
NUM2 Keypad 2
NUM3 Keypad 3
NUM4 Keypad 4
NUM5 Keypad 5
NUM6 Keypad 6
NUM7 Keypad 7
NUM8 Keypad 8
NUM9 Keypad 9
NUM0 Keypad 0
NUM* Keypad *
NUM/ Keypad /
NUM+ Keypad +
NUM- Keypad -
NUM. Keypad .

_SPACE Space bar (must always be shifted - see below)

All letter and number keys i.e.

A to Z and 0 to 9 referred to by letter/number alone.

Shifted keys

Any of the above prefixed with any combination of 'S' for shift, 'C' for control or 'A' for alt. Note that in windows, the right hand ALT key performs the same action as CONTROL-ALT.

Notes

Unshifted letter and number key definitions will not function when a text edit window such as the simulator command window (F11) is active. Space bar definitions must always be shifted.

The same codes can be used for menu short cuts. See “[DefMenu](#)” on page 368

Key definition will be lost when SIMetrix exits. To make a key or menu definition permanent you can place the command to define it in the startup script. To do this, select command shell menu File|Scripts|Edit Startup and add the line above.

Examples

To define control-R to place a resistor on the schematic sheet, enter the command:

```
DefKey CR "inst res" 4
```

The built in definition for F12 to zoom out a schematic is

```
DefKey F12:SCHEM "zoom out" 4
```

This definition only functions when a schematic is active. A similar definition for F12:GRAPH zooms out a graph when a graph window is active.

DefMenu

```
DefMenu [/comgroup command_group] [/submenu] [/id id] [/pos pos]  
[/features features] [/shortcut key_code] [/noRepeat] [/immediate]  
[/insert] menuname command_string when_to_enable
```

Defines custom menu. Supersedes DefItem.

<i>command_group</i>	This can be used with the function GetLastCommand (page 172). GetLastCommand returns the text of the most recent command executed which specified the supplied command group value. The command DefButton (page 361) also uses this feature
<i>/submenu</i>	If specified, an empty submenu will be created as long as it does not already exist.
<i>id</i>	This item is used by the edit menu GUI. It is not needed for regular use
<i>pos</i>	Position of menu. '1' means the top position. If omitted, the menu is placed at the bottom.

<i>features</i>	List of license features that must be available for the menu definition to be accepted. This is used to build the standard menu system and is not usually required										
<i>/shortcut key_code</i>	Specify key or key combination to activate menu. Key description is placed on right hand side of menu item. For list of possible values see “DefKey” on page 365, but note that key pad keys (e.g. NUM1, NUM* etc.) cannot be assigned as menu shortcuts. Also note that DefKey has precedence in the event of the key or key combination being defined by both DefKey and DefMenu.										
<i>/norepeat</i>	Do not save menu action in “repeat last menu” buffer. This must be used for any menu that recalls a previously executed menu										
<i>/immediate</i>	Immediate mode. Command is executed immediately even if another operation - such as a simulation run or schematic editing operation - is currently in progress. For other options the command is not executed until the current operation is completed. Only a few commands can be assigned with this option. These are Abort AbortSIMPLIS Cancel DefMenu DefKey Echo Let Move Pan Pause Quit RotInst Select ScriptAbort ScriptPause ScriptResume Shell Wire Zoom										
<i>menuname</i>	Composed of strings separated by pipe symbol: ' '. First name must be one of the following. Note that these are case-sensitive under Linux.: <table border="0" style="margin-left: 20px;"> <tr> <td>Shell</td> <td>Command shell menu</td> </tr> <tr> <td>Schem</td> <td>Schematic popup menu</td> </tr> <tr> <td>Simetrix</td> <td>Schematic popup menu - SIMetrix mode only</td> </tr> <tr> <td>Simplis</td> <td>schematic popup menu - SIMPLIS mode only</td> </tr> <tr> <td>Graph</td> <td>Graph popup menu</td> </tr> </table>	Shell	Command shell menu	Schem	Schematic popup menu	Simetrix	Schematic popup menu - SIMetrix mode only	Simplis	schematic popup menu - SIMPLIS mode only	Graph	Graph popup menu
Shell	Command shell menu										
Schem	Schematic popup menu										
Simetrix	Schematic popup menu - SIMetrix mode only										
Simplis	schematic popup menu - SIMPLIS mode only										
Graph	Graph popup menu										

GraphMain	Graph fixed menu
SchemMain	Schematic main menu
SimetrixMain	Schematic main menu - SIMetrix mode only
SimplisMain	Schematic main menu - SIMPLIS mode only
Symbol	Symbol editor popup menu
SymbolMain	Symbol editor fixed menu

For Shell, SchemMain, SimetrixMain, SimplisMain, GraphMain and SymbolMain menus, this must be followed by two or more names separated by '|'. The first is the menu name as it appears on the menu bar. The second can be the name of a menu item (which is actioned when selected) or a sub menu containing menu items or further sub menus. Sub menus can be nested to any level.

Use the '&' symbol to define an underlined ALT-key access letter.

Graph, Schem, Simetrix, Simplis must be followed by at least one name. Sub menus may also be defined for these.

To define a menu separator use the item text '-'
Note that if any of the menu name contains spaces it must be enclosed in quotation marks.

See examples below.

when_to_enable

A Boolean expression specifying under what circumstances the menu should be enabled. (The menu text turns grey when disabled). If omitted the menu will always be enabled. The expression may contain the following values.

SchemOpen	TRUE when there is at least one schematic open.
InstSelected	TRUE when at least one component is selected on the selected schematic
Selected	TRUE when at least one component or at least one wire is selected on the current schematic
PropertiesSelected	TRUE if schematic properties are selected
ClipboardEmpty	TRUE if there is no schematic clipboard data available
SimPaused	TRUE when the simulator has been paused.
SimRunning	TRUE when the simulator is running.
CircuitLoaded	TRUE when a circuit has been loaded to the

simulator. (This happens when ever a simulation is run. A circuit can be unloaded with the Reset command).

GraphOpen	TRUE when there is at least one graph window open.
GraphCursorOn	TRUE when graph cursors are switched on
GraphObjectSelected	TRUE if any graph annotation object, such as a legend box, is currently selected.
CurvesSelected	TRUE if any curves are selected
LiveMode	TRUE when a command has not completed.
Never	Always FALSE i.e menu permanently disabled.

These values can be combined with the operators:

&& logical AND

|| logical OR

== equals

!= not equal

! NOT

Parentheses may also be used.

Note that this expression is not related to vector expressions or the expressions that can be used in netlists or the command line.

Notes

You can use DefMenu to redefine an existing menu. In this situation the position of the menu will not change but the command it executes and any shortcut key can be altered. Note that *menuname* is case-sensitive under Linux. So if you define a menu that differs only by case from an existing menu, a new menu will be created on Linux, but on Windows the existing menu will be modified. This allows filenames to be used for menu names.

Note that it isn't possible to add or remove a top level main menu definition while the window is open. For schematic, graph and symbol editor windows, this means that the definition of a new top level menu will not take effect until the windows are closed and reopened. For the command shell, top level main menu definitions can only be made in the startup script which runs before the command shell is visible.

This restriction only applies to the top level menu, that is the menu name that is permanently visible in the menu bar. Menu items and sub menus under the top level menu can be added, removed and redefined at will.

Examples

The following are definitions for some of the standard menus. Definitions for all the standard menus can be found on the install CD in the Scripts folder. (A CD image may be downloaded from our web site if you do not have the physical CD)

Change value schematic popup menu by calling the *value* script. (Note this must be entered on one line)

```
DefMenu "Schem|Change &Value" "value /ne"  
"InstSelected&&!LiveMode"
```

Separator in schematic popup

```
DefMenu "Schem|-"
```

Graph popup to enable cursors

```
DefMenu "Graph|Cursors &On" "cursormode /ne on" "!LiveMode"
```

Del

Del *filename*

Deletes the specified file. Wildcards may be used for *filename* e.g. *.*. '*' matches any sequence of zero or more characters. '?' matches a single character. Any file matching the specification will be deleted.

DelCrv

DelCrv *curve_id* ... |*curve name* ...

Deletes the specified curve or curves on the selected graph. *curve_id* is returned by the functions [GetSelectedCurves](#) (page 187), [GetAxisCurves](#) (page 146) and [GetAllCurves](#) (page 143).

Optionally a curve name may be specified. This must be the whole text of the curve legend. It is the value returned by the function [GetCurves](#) (page 152).

Delete

Delete

Deletes the currently selected components and/or wires in the selected schematic sheet

DeleteAxis

DeleteAxis *axis_id*

Deletes the specified axis.

axis_id Axis id as returned by functions [GetAllYAxes](#) (page 144), [GetSelectedYAxis](#) (page 188) or [GetCurveAxis](#) (page 152).

Note that an axis may only be deleted if it is empty i.e. has no attached curves. Also the main axis may not be deleted.

DeleteGraphAnno

DeleteGraphAnno *object-id*

Deletes a graph annotation object such as a curve marker or legend box. See “[Graph Objects](#)” on page 448 for details on graph annotation objects.

object-id Id of object to be deleted.

DeleteSymbolProperty

DeleteSymbolProperty *property_name*

Deletes the specified property from a symbol editor symbol.

property_name Name of property to be deleted. The command will yield an error if this is omitted. If a property of that name is not found, no action will be taken.

DelGroup

DelGroup [/all] [/cleanup] [/nodelete] *groupname2* ...

/all If specified all groups except the user group are destroyed.

/noDelete Inhibits delete of associated temporary data file. This file will only be deleted any way if the option variable DataGroupDelete is set to OnDelete.

/cleanup Specify this switch if the associated data file is going to be reused as it may speed up the read operation especially if the data was created by a simulation that was paused. If the file will be deleted then this switch has no benefit but will do no harm other than to slow the execution of this command a little.

Deletes specified groups. See “[Groups](#)” on page 37 for more information.

See Also

“[CreateGroup](#)” on page 354
Function “[Groups](#)” on page 210

DelLegendProp

DelLegendProp *curve_id* *property_name*

curve_id Id of curve which possesses property. Curve id is returned by the functions GetSelectedCurves ([page 187](#)), GetAxisCurves ([page 146](#)) and GetAllCurves ([page 143](#))

property_name Name of property to be deleted. The function

GetLegendProperties ([page 173](#)) returns legend properties owned by a specified curve.

DelMenu

DelMenu [*/bypos position*] [*/force*] [*/keepid*] *menuname*

Deletes specified menu item, or submenu.

<i>position</i>	The menu to be deleted is identified by its position. The first item in the menu is at position zero.
<i>/force</i>	If specified, will allow complete submenus to be deleted. Otherwise this command will only delete a single menu item
<i>/keepid</i>	Do not delete the action item that forms part of the menu definition. This is used by the edit menu GUI and is not usually required for normal use

menuname Composed of strings separated by pipe symbol: '|'. First name must be one of the following:

SHELL	Command shell menu
SCHEM	Schematic popup menu
GRAPH	Graph popup menu
LEGEND	Popup menu in graph “legend panel”
SCHEMMAIN	Schematic main menu
SYMBOL	Symbol editor popup menu
SYMBOLMAIN	Symbol editor fixed menu

The remaining strings identify the menu and item names. See [“DefMenu” on page 368](#) for details on menu names.

DelProp

DelProp *property_name*

Delete specified property from selected schematic instances.

property_name Name of property to be deleted

DelSym

DelSym *symbol_name*

Deletes a schematic symbol from the global library or from the current schematic.

See Also

[“Schematic Symbol Script Definition” on page 442](#)

DelSymLib

DelSymLib

This command is now obsolete.

Detach

Detach

Unselects partially selected wires on schematic. A partially selected wire is one which is selected at one end only. Executing this command immediately prior to a move operation effectively disables 'rubberbanding'.

Discard

Discard [/vec *vecname*] | [*groupname*]

Frees up memory used for vectors. This does not destroy the vectors, just removes any copies that reside in RAM. The data is always stored on disc and can be recovered to RAM when needed.

/vec	If specified <i>vecname</i> specifies a single vector.
<i>groupname</i>	Name of group data is to be discarded. Use current group if omitted.

Notes

It is rare that this command is needed but may be useful if you are running long simulations and the data generated is so large that a great deal of disk swapping is taking place.

The vectors created by the simulator are initially stored in a file. If they are needed - usually for plotting a graph - the data is copied to memory. Once the data has been copied to memory, it will stay there until the group to which the vector belongs is destroyed. Simply closing the graph that used the data will not free up the memory as it is assumed that the data may be needed again and the process of reading from the disk can be time consuming. If the data is very large it will consume a lot of memory which can have adverse consequences.

The discard command deletes the data stored in memory for all vectors in the specified group or a single vector if /vec is specified. It does *not* delete the vectors altogether as they are still stored on disc in the temporary file. After discarding a group, it is still possible to plot all vectors that it contains.

Display

Display [/file *file*] [/append *append_file*] [/notype] [/notitle] [/type *type*]
[*groupname1* [*groupname2* ...]]

Script Reference Manual

Displays list of all vectors in specified groups or current group by default. Lists the name, physical type (e.g. voltage, current etc.) data type (real, complex, string, alias) and size of each vector

<i>file</i>	Output result to <i>file</i>
<i>append_file</i>	Append result to <i>append_file</i>
<i>/notype</i>	Do not list the data type
<i>/notitle</i>	Do not display the header showing the group name
<i>type</i>	Filter result according to <i>type</i> . <i>type</i> is a list of typenames separated by ' '. Possible values are: real - real values complex - complex values string - string values alias - alias values

See Also

[“Expressions” on page 30](#)

DrawArc

DrawArc [*theta* [*v_over_h*]]

Initiates “arc draw” mode in the currently open symbol editor. This is an interactive command.

<i>theta</i>	Swept angle in degrees (integer). Default = 90
<i>v_over_h</i>	Vertical radius/Horizontal radius. Default = 1 (circle)

DrawPin

DrawPin [*/forcerepeat*] [*base_name*]

Initiates “pin draw” mode in the currently open symbol editor. In this mode a pin symbol is presented for the user to place at the desired location on the symbol sheet.

<i>/forcerepeat</i>	If specified, the operation will be repeated until the user cancels with the right mouse button. Each new pin will be named according to the base name appended with an integer to make it unique.
<i>base_name</i>	Name of pin. If a pin of that name is already present on the schematic, the name will be appended with a number to make it unique. If the base name is already appended with a number, that number will be incremented until an unused name is found.

Echo

Echo [*/file/append filename/handle handle*] */page /box text*

Echoes *text* to the message window or to a file

<i>/file filename</i>	If present <i>text</i> is output to <i>filename</i> . If <i>filename</i> exists, it is overwritten
<i>/append filename</i>	If present <i>text</i> is appended to <i>filename</i> . If <i>filename</i> does not exist, it is created.
<i>/handle handle</i>	File handle as returned by the function OpenFile (page 247) . Text will output the file referenced by this handle.
<i>/page</i>	Prefixes output with a ASCII form feed character
<i>/box</i>	Text is output inside a box composed of asterix characters. This is useful for titles and headings. Currently only works correctly when used with <i>/file</i> or <i>/append</i> .

EditColour

EditColour *colour-name colour-spec*

Changes the spec for the named colour object.

<i>colour-name</i>	Name of colour object. This can be any of the names returned by the <code>GetColours()</code> function. (These are listed when the menu <code>File Options Colour...</code> is selected.)
<i>colour-spec</i>	Text string that defines the colour. The functions <code>GetColourSpec()</code> and <code>SelectColourDialog()</code> return colour spec values.

EditCopy

EditCopy [*/mono*]

Copies selected schematic items to clipboard for pasting to SIMetrix and other applications.

<i>/mono</i>	If specified, the image obtained when pasting to other applications will be monochromatic. This switch has no effect when pasting to SIMetrix windows.
--------------	--

The EditCopy - in conjunction with EditPaste - make it possible to copy blocks of schematic from one schematic window to another.

The EditCopy commands differs from the older CopyClipSchem command in that only selected items are copied. Further, schematics copied with CopyClipSchem can only be pasted into other applications.

See Also

[“EditPaste” on page 378](#)

[“CopyClipSchem” on page 352](#)

EditCut

EditCut

Equivalent to the sequence:

Detach
EditCopy
Delete

Deletes selected components and places them in the clipboard

EditFile

EditFile *filename*

Opens an external text editor to edit specified file. The path of the text editor may be specified by the ‘Editor’ option which may be set in the File Locations tab of the options dialog box. (Menu File|Options|General...). This is NOTEPAD by default.

EditFont

EditFont *font-name font-spec*

Changes the spec for the named font object.

<i>font-name</i>	Name of font object. This can be any of the names returned by the GetFonts() function. (These are listed when the menu File Options Font... is selected.)
<i>font-spec</i>	Text string that defines the font. The functions GetFontSpec page 163 and SelectFontDialog page 286 return font spec values.

EditPaste

EditPaste

Pastes items from clipboard to a schematic sheet. Only items copied by SIMetrix (using EditCopy command) may be pasted.

EditPin

[/name *new-pin-name*] *symbol-name pin-number*

Edit a pin name of a symbol in the currently installed symbol library.

<i>new-pin-name</i>	New pin name for symbol pin. This may not contain spaces
<i>symbol-name</i>	Internal name of symbol owning the pin to be edited
<i>pin-number</i>	Number of pin to be edited.

EndSym

EndSym [/tri]

EndSym is a Symbol Definition Command. All symbol definitions must end with this command and begin with CreateSym.

/tri If specified, generated symbols will have a triangular shape. This is only valid for symbols with pins specified using PinDef command.

See Also

[“Schematic Symbol Script Definition” on page 442](#)

ExecuteMenu

ExecuteMenu *menu-item*

Executes the specified menu.

menu-item Menu definition as described in DefMenu command.

Execute

Execute [/echo] [/literal] *command*

Run the script or command *command*.

/echo Command is copied to the command history drop down box in the command shell

/literal This is best explained with an example. Both of the following will do the same thing:

Execute /literal "inst npn"

Execute inst npn

But this will throw an error:

Execute "inst npn"

The problem with the last example is that the Execute command interprets the first token in *command* as the actual command or script name and the remainder of *command* as the arguments to it. Because "inst npn" is enclosed in quotation marks, it is treated as a single item specifying the command name "inst npn" which is incorrect.

Always use this switch if the complete command with any arguments is stored in a variable to be accessed by a braced substitution. E.g.

```
Let command = 'inst npn'  
Execute /literal {command}
```

command Command to be executed with arguments if required. See /literal above for more information.

Scripts are usually run by simply entering their name in the same way as a command is entered. However, the script is executed slightly differently if run using the Execute command. If a script is called from another script in the normal way, the called script is read in and parsed before the main script is executed. If the Execute command is used, the called script is not read in until and unless the Execute command is actually executed. This has two main applications.

1. The name of the called script is not known initially. This is the case with the menu item File|Script|Run Script... The script name is selected from a dialog box. The Execute command is used to implement this menu item.
2. The called script is very long and is not always called by the calling script. It may take some time to read in and parse the called script. This time would be wasted if the script is not actually called.

Avoid using Execute if a script is called within a loop. The script would be read in and parsed each time around the loop which is very inefficient.

Focus

Focus [/named *window_name*] [/userid *user_index*] [*schem|graph*]

window_name If specified the window of the given name will be given input focus. The name of the window is the text in the title bar with '(Selected)' stripped off. Window name is also returned by the function [GetWindowNames \(page 208\)](#), but to focus a specified window, use the *user_index* instead. See next...

user_index User index as returned by the function [GetWindowNames \(page 208\)](#)

schem|graph Currently selected schematic or graph window receives input focus.

Only one of the above items can be used at a time. If multiple specifications are provided, *window_name* takes priority over *user_id* which takes priority over *schem|graph*.

See Also

[GetWindowNames \(page 208\)](#)

FocusShell

FocusShell

Selects the Command Shell and assigns it keyboard focus.

GraphZoomMode

GraphZoomMode X|Y

Specifies mode of next mouse zoom operation

X Only X axis will be zoomed

Y Only Y axis will be zoomed

All subsequent zoom operations will be applied to both axes.

Help

Help [/file *filename*] [/contents] [/context *context_id*] [*topic*]

Opens the SIMetrix help system.

<i>filename</i>	If specified, help will be obtained from <i>filename</i> . Otherwise help file will be SIMetrix.chm
/contents	Included only for backward compatibility. "Help" and "Help /contents" both perform the identical action. But /contents will override /context and <i>topic</i> to retain earlier behaviour.
<i>context_id</i>	Opens specific topic identified by an integer. This is used by some internal scripts but is not supported for user application.
<i>topic</i>	If specified, help system will display page relating to <i>topic</i> . If <i>topic</i> does not exist, a list of available topics will be displayed.

Example

To display help on the .TRAN simulator directive type:

```
Help .tran
```

HideCurve

HideCurve *curve_id*

Hides specified curve

curve_id Id of curve to hide. Curve id is returned by the functions GetSelectedCurves ([page 187](#)), GetAxisCurves ([page 146](#)) and GetAllCurves ([page 143](#))

See Also

ShowCurve [page 427](#)

HighlightCurve

HighlightCurve [/clear | /unique] *curveId*

Script Reference Manual

Highlights the selected curve. A curve is highlighted by displaying it in a brighter colour and bringing it to the top - i.e. it is drawn last. Also, highlighted curves are displayed in increased thickness, the amount determined by the HighlightIncrement option setting.

<i>curveId</i>	Id of curve to be highlighted (or unhighlighted if /clear specified)
/clear	The specified curve will be unhighlighted.
/unique	The specified curve will be highlighted and all others will be unhighlighted.

Hint

Hint [/help *help_context_id*] [/id *id*] [/icon *icontype*] *message*

Displays a message box intended to be used to provide hints to the user. The box contains a check box allowing the user to choose not to receive such hints again.

help_context_id If specified, the box will show a help button which will display the help topic specified by *help_context_id*. This is used in some internal scripts but has limited user application.

id Identifier used to identify hint for the purposes of saving the redisplay status controlled by the “Don't show this message again” check box . If not supplied, a default will be used derived from the message text. This is satisfactory in most cases and there is rarely ever a need to use this switch.

icon_type Controls the icon displayed in the hint box. This may be one of:

info - An icon showing the letter ‘i’ indicating that this message is for information only. This is the default

warn - An icon showing an exclamation mark in a yellow triangle indicating that the message is a warning

error - An icon showing a cross in a red background indicating an error condition. This is usually inappropriate for a hint, but is included for completeness

question - An icon showing a question mark indicating a question. Currently the hint box is not interactive so the usefulness of this is limited

message Message to be displayed.

HourGlass

HourGlass [/clear]

Displays the hourglass cursor shape indicating that some action is in progress. The normal cursor is automatically restored when control returns to the command line.

`/clear` Returns cursor to normal. HourGlass maintains a count of the number of times it is called and in order to release the cursor, it must be called an equal number of times with the `/clear` switch specified.

ImportSymbol

`ImportSymbol [/loc x y] [/local] [/path pathname] [/comp] name`

Imports an existing symbol to the currently open symbol editor sheet.

`x, y` If `/loc` switch specified, the symbol is placed at the location specified by `x` and `y`. In practice this location may only be used in a relative manner as the exact location on the symbol sheet of the origin will be adjusted to ensure that the symbol is in view.

`/local` The symbol will be obtained from the local library of the current schematic. If not specified the symbol will be obtained from the global library.

`/path pathname` If specified, the symbol will be converted to a component to be saved in the file specified by *pathname*.

`/comp` Opens the symbol for a component whose path is specified by *name*.

name Symbol name.

Note

If the current symbol sheet is empty, the named symbol will become the current symbol in that sheet. This will be reflected in the caption bar text and the default symbol to be saved when File|Save... is selected.

Inst

`Inst [/centre] [/loc x y orient] [/select] [/repeat] [/norepeat] [/orient orient] [/comp] [/nolocal] [/useph] symbolname [propname propvalue]`

`/centre` If specified the cursor will be positioned in the centre of the selected schematic window. Otherwise the cursor will remain at whatever position it happens to occupy when the command is executed.

`/loc x y orient` If specified, instance is placed directly on sheet without user interaction at the location specified by `x` and `y` and orientation specified by *orient*. These values are relative. The origin of the schematic is not fixed. Usually the values used would have been returned from a call to the function `InstPoints` ([page 220](#))

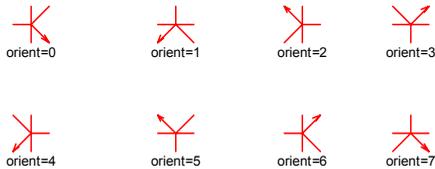
`/select` If specified, the instance is selected after being placed on the schematic.

`/repeat` If specified the instantiation is repetitive. This means that once one instance has been placed, another will be presented. This

continues until the user presses the right mouse key. This switch will be ignored if the RepeatPlace option is set to 'Never' (Placement options set to 'Never' in schematic sheet of options dialog). If RepeatPlace is set to 'Always', the repeat action will be enabled even if this switch is not present as long as /norepeat isn't present. If the /loc switch is present repeat action is disabled in all circumstances.

/norepeat If specified a single instance will be placed regardless of the value of the 'RepeatPlace' global option

/orient orient Specifies orientation of symbol. Value from 0 - 7 as illustrated below.



/comp Places a component symbol whose path is specified by *symbolname*.

/nolocal Only effective if /comp also specified. Inhibits search of the schematic's local symbol library for an existing copy of the symbol.

/useph Only effective if /comp also specified. Will place a place holder symbol if the symbol specified is not found

symbolname Name of symbol.

propname If specified, property of this name is changed to *propvalue*

propvalue See above

Places an instance of *symbolname* onto current schematic. User must press left mouse key to fix symbol to schematic.

KeepGroup

KeepGroup on/off

Switches *keep status* of current group.

Groups generated by the simulator start with their *keep status* set to off. This means that it will automatically be deleted when a certain number (set by the GroupPersistence option) of new groups are created. If the *keep status* is set to on then automatic deletion is disabled. Groups read from a file using OpenGroup start with their *keep status* set to on.

Let

Let [*vector_expression*]

Evaluates a vector expression.

vector_expression vector expression to be evaluated. Information on vector expressions can be found in “Expressions” on page 30

To be meaningful *vector_expression* must contain the assignment operator '='.

If *vector_expression* is omitted a list of vectors in the current group will be displayed.

Examples

Create a new vector of name power:

```
let power = r1#p*(r1_p-r1_n)
```

Listing

Listing [/file *filename*] [/errors] [/append *filename*]

Displays or outputs to a file a listing of the current netlist.

/file *filename* Result is written to file of name *filename*

/append *filename* Result is appended to file of name *filename*

/errors Only lines that are in error are output

Note the *current netlist* is the netlist for the circuit most recently run or checked. It will include all models and subcircuits pulled in from libraries.

ListModels

ListModels *filename*

Generates a dictionary of all models and subcircuits currently available to the simulator (i.e. installed with menu File|Model Libraries|Add/Remove Libraries see *User's Manual* for details). Result is written to *filename*. A single line will be produced for each model or subcircuit found containing the device name, its type (NPN, JFET, subcircuit etc.) and the filename in which it was found along with the line number.

ListOptions

ListOptions *filename*

List all global options to file. Global options are set using the command [Set](#) (page 420)

filename File to receive options

Listing contains one line per option with each line being a semi-colon delimited list in the following form:

name;type,default_value

where:

name Name of option

type Type of option. One of 'bool', 'string' or 'real'

default_value Default value if not set, or if unset using command [UnSet \(page 430\)](#)

ListStdButtonDefs

ListStdButtonDefs [*filename*]

Lists the built in toolbar button definitions. These are in the form of the DefButton command used to create the definition.

filename If specified, the results will be written to *filename*. Otherwise the results will be displayed in the command shell.

ListStdKeys

ListStdKeys *filename*

Writes built in key definitions to *filename*.

LoadModelIndex

LoadModelIndex

Forces model library indexes to be re-checked and loaded. Model library indexes are binary files that allow the rapid location of simulation models. When SIMetrix starts, it checks that the indexes are up to date by comparing file dates. If any files have been changed, the appropriate index file will be rebuilt. When this process is complete, the indexes are read in to memory for fast access.

This command forces SIMetrix to repeat the above procedure. This may be necessary if additional files are added to a directory where models reside while SIMetrix is running. SIMetrix can usually detect this automatically if the drive is local but cannot always do so for network drives.

Note the menu Model Library|Rebuild Catalog calls this command.

The work of reloading indexes is actually performed by the simulator in the background so this command returns immediately even though the process can take several seconds. If you start a simulation immediately after executing this command, there will be a pause until the reload is complete.

MakeAlias

MakeAlias *variable*

Converts a string variable to an alias.

variable variable to be converted

An alias is a string representing a numeric expression. For more information see [“Aliases” on page 33](#)

MakeCatalog

MakeCatalog *outfile_name* *main_catalog* [*user_catalog*]

This command builds a catalog file for use by the parts browser. This is normally called OUT.CAT and resides in the SCRIPT directory.

outfile_name File name for catalog. This must be OUT.CAT for use with browser

main_catalog Main database of parts. This would usually be ALL.CAT which resides in the SIMetrix root directory

user_catalog User database of parts. This would usually be called USER.CAT which resides in the script directory

The MakeCatalog command is one of the components of the Parts Browser system. The parts browser requires a catalog file which lists all the models available to the simulator and for each provides the name of a suitable schematic symbol, a category, pin mapping if relevant, a symbol model property (e.g. X for subcircuits, Q for BJTs) and a preferred pathname if there is more than one model of that name. The MakeCatalog command builds this catalog using the data files *main_catalog* and *user_catalog* to obtain information about known parts.

MakeSymbolScript

MakeSymbolScript [/all] [/catalog *catalog_name*] [/append]
[/sortProps] *filename* [*symbol_name* ...]

Creates a script definition of a symbol or group of symbols. For details of script definitions see [“Schematic Symbol Script Definition” on page 442](#).

/all If specified, scripts for all symbols in the global library will be created.

catalog_name If specified, scripts for all symbols in the specified catalog of the global library will be created. This overrides /all

/append Result will be appended to specified file

/sortProps If specified, all visible properties are ordered alphabetically in the output script. Properties are defined with the AddProp command

filename Path of file to be written.

symbol_name Name of symbol to be scripted. Any number may be specified. If /all or /catalog are specified, this argument will be ignored. If they are not this argument becomes compulsory.

MakeTree

MakeTree *path*

Creates the specified directory path. Unlike the MD command, MakeTree will create any subdirectories required to make the whole path.

MCD

Mcd *directory_name*

Makes a directory and sets it as current. (Same as Md followed by Cd)

directory_name Name of directory to be created.

MD

Md *directory_name*

Creates a new directory. Md is similar to the DOS MD and MKDIR commands and the Linux mkdir command.

directory_name Name of directory to be created.

Message

Message [*text*]

Displays a message in the status window of the currently selected schematic. This will temporarily overwrite status information at the base of the schematic until Message is called with no arguments.

*text*Text to be displayed. If omitted, status window returns to normal view.

MessageBox

MessageBox *text* [*caption*]

Displays message box with text *text* and caption *caption*. Note that there is also the function [MessageBox \(page 236\)](#) which is more flexible.

Move

Move [/mode *mode*]

Initiates the schematic move operation. User interactive command.

mode Specifies editing mode to use for move operation. Options are:
“default” - Use option setting “SchematicMoveMode”
“ClassicMove” - Basic rubberbanding mode
“GrowWire” - As ClassicMove but enables creation of a wire between connected pins

“Orthogonal” - Wires edited so that they remain at right angles as much as possible

MoveCurve

MoveCurve *curve_id axis_id*

Moves a curve to a new y-axis

<i>curve_id</i>	Id of curve as returned by Curve id is returned by the functions GetSelectedCurves (page 187), GetAxisCurves (page 146) and GetAllCurves (page 143)
<i>axis_id</i>	Axis id as returned by functions GetAllYAxes (page 144), GetSelectedYAxis (page 188) or GetCurveAxis (page 152)

MoveFile

MoveFile [/force] *path-1 path-2*

Moves *path-1* to *path-2*.

<i>/force</i>	If specified, <i>path-2</i> will be overwritten if it already exists. If not specified, the command will fail if <i>path-2</i> exists.
---------------	--

MoveMenu

MoveMenu [/bypos *position*] *menupath shift-by*

Moves the position of a menu item by a specified count.

<i>position</i>	Optional number that identifies a menu item by its position within a sub-menu. If this is specified the <i>menupath</i> must identify a sub-menu rather than a menu item.
<i>menupath</i>	Menu path. See “DefMenu” on page 368 for full details
<i>shift-by</i>	Number of positions by which menu is moved. A positive number moves the menu down, a negative number moves it up.

MoveProperty

MoveProperty *property-name*

This is an interactive command. It switches the schematic editor into ‘move property’ mode. In this mode the user can move the specified property for all selected instances. The mode is completed by pressing the left or right mouse key. The left key will fix the new property position and the right key will cancel the mode and leave the properties unmodified.

Note

In SIMetrix, property positions can be defined in one of two ways namely ‘Auto’ and ‘Absolute’. Most of the standard symbols have their properties defined as ‘Auto’. This

means that SIMetrix chooses the location of the property on a specified edge of the symbol and ensures that it doesn't clash with other properties on the same edge. 'Auto' properties are always horizontal and therefore easily readable. The position of 'Absolute' properties is fixed relative to the symbol body regardless of the orientation of the symbol and location of other properties. When the symbol is rotated through 90 degrees, absolute text will also rotate.

When a visible property on a symbol is moved using the MoveProperty command, it and all other visible properties on that symbol are converted to 'Absolute' locations. This is the only way that the positions of all properties can be preserved.

Netlist

```
Netlist [/num] [/subckt subcircuit_name] [/nopinnames] [/nooutput]
[/diag full|partial] [/template templateProps] [/sep separator]
[/top]/[plain] [/lang language] [wireTemplate wire_template] [/dotEnd]
[/noDescend]/[f11top] [/nodemap] [/simplis] [/path path] [/sort]
[filename ]
```

Generates a netlist for the currently selected schematic. The netlist command also assigns names to schematic nets. If the schematic contains hierarchical blocks, their underlying schematics will also be netlisted and included in the main netlist as subcircuits.

<code>/subckt <i>subcircuit_name</i></code>	If specified, circuit is netlisted as subcircuit. In this case the netlist is enclosed with a <code>.subckt</code> control at the beginning and a <code>.ends</code> control at the end.
<code>/num</code>	If specified, a SPICE 2 compatible netlist using node numbers is created.
<code>/nopinnames</code>	If specified, the <code>pinnames</code> specifier is not output for X devices. The <code>pinnames</code> specifier is proprietary to SIMetrix and is not supported by other simulators. Use this option if you are creating the netlist for another purpose e.g. to input to an LVS program.
<code>/nooutput</code>	If specified, no netlist output is generated. The net names attached to wires are updated.
<code>/diag full partial</code>	If specified, a diagnostic report will be produced. This details: Implicit node connections (using terminal symbol). Bus name translations. These occur if two buses with different names are connected Dangling wires and unused device pins. If the <code>diag</code> is set to <code>partial</code> , only dangling wires

	and pins are reported.
<i>/template <code>templateProps</code></i>	Property names to be used as templates. A template is a string that specifies a format to be used for the netlist line for the device that owns it. By default the template property name is "TEMPLATE". This can be overridden with this switch. Multiple template property names may be specified by separating them with a pipe symbol (' '). See the description of the template property in the "User's Manual".
<i>/sep <code>separator</code></i>	May be a single character or "none". Default is '\$'. To comply with SPICE syntax each device line starts with a letter that identifies the type of device. Usually this letter is determined by the MODEL property. If the component reference of the device does not begin with the correct letter it is prefixed with the correct letter followed by the character specified by this option.
<i>/top</i>	For hierarchical schematics, the line ".KEEP /subs" is automatically output to tell the simulator to output data for all subcircuits. Specifying this switch inhibits this action thus restricting data output to the top level.
<i>/lang <code>language</code></i>	Name of language to be output at the top of the netlist output. This is in the form " <i>*#language</i> " and is used by SIMetrix for compatibility with other simulators. Default is "SIMETRIX"
<i>/wireTemplate <code>wire_template</code></i>	Format for bus wires. <i>wire_template</i> may contain the keywords %BUSNAME% and %WIRENUM%. These resolve to the bus name and wire number respectively. So a spec set to %BUSNAME%#%WIRENUM% would give the default, i.e. bus names like BUS1#2. A spec of %busname%[%wirenum%] would give bus names like BUS1[2].
<i>/dotEnd</i>	Forces .END to be placed at the end of the netlist
<i>/noDescend</i>	Netlister does not descend into hierarchy and processes items at the top level only.
<i>/f1ITop</i>	The contents of the F11 window are placed before the netlist lines generated by the schematic. Otherwise they are placed after the schematic netlist lines.
<i>/nodemap</i>	Generates SIMPLIS .NODE_MAP controls for user named nets.
<i>/plain</i>	Equivalent to /noPinnames /top /lang none

<code>/simplis</code>	Specify this if creating a netlist for use with SIMPLIS. Forces switches: “/dotEnd /flITop /nodemap /num /nopinnames /sort”. If /template is not specified, a default of “/template simplis_template/template” will be forced. Finally if /wireTemplate is not specified, a default of “/wireTemplate %busname%%\$%wirenum%” will be forced.
<code>/path path</code>	If specified, the netlist operation will be performed on the schematic at the specified file system path. If the specified schematic is currently open, the netlist generated will reflect the displayed version rather than the contents of the file.
<code>/sort</code>	If specified, the netlist lines will be output in alphanumeric sorted order.
<code>filename</code>	File to which netlist is written. If not specified, the netlist is displayed in the message window.

NewAxis

NewAxis

Creates a new y-axis. This will be initially empty and selected. See User's manual for more information on multiple y-axes.

NewGraphWindow

NewGraphWindow

Creates a new graph window to which new graphs may be directed.

NewGrid

NewGrid

Creates a new grid. See User's manual for more information on axes and grids.

NewPrinterPage

NewPrinterPage

Advances printer to the a new page. This may be used for customised or non-interactive printing. See “[Non-interactive and Customised Printing](#)” on page 463.

NewSchem

NewSchem [/newWindow] [/simulator *simulator*] [*filename*]

Creates a new schematic sheet within the currently selected schematic window. If no schematic window is open, one will be created.

<i>filename</i>	Name of file to which schematic will be saved when save command is issued or when the menu File Save is selected. Note that <i>filename</i> is not created by this command.
<i>simulator</i>	Specifies initial simulator mode. Set to 'SIMPLIS' to open an empty schematic switched to SIMPLIS mode or 'SIMetrix' to open in SIMetrix mode. If not specified, the schematic will open in a mode determined by the 'InitSchematicSimulator' option setting. (Defined using command "Set" on page 420)
/newWindow	If specified, a new schematic window will be created.

NewSymbol

NewSymbol

Opens a new symbol editor sheet.

NoPaint

NoPaint

This command has no effect unless executed from within a script. It inhibits all updates to graphs until script execution is complete. This is useful when a number of operations are performed on a graph. By calling this command at the start of a script, multiple graph operations can be performed much faster and more smoothly.

NoUndo

NoUndo [/nocapture] [/release]

Inhibits saving to undo buffer until command returns to the command line. This allows multiple operation to be treated as one for the purposes of the Undo feature. For example, suppose you have a script that edits a number of schematic instances. Normally, if you run the script then select Undo, only the most recent change will be undone. The user would need to select Undo many times to return the circuit to the state before the script was run. If NoUndo is called at the start of the script, Undo will return the schematic to the start state in a single operation.

/nocapture	Normally NoUndo, saves the current state so that the next undo operation will restore the state to immediately before NoUndo was called. The /nocapture switch inhibits this.
/release	Restores undo buffer save operations. This happens automatically when control returns to the command line.

OpenGraph

OpenGraph [/newwindow] *filename*

Opens the graph file *filename* and displays.

/newwindow	Of specified, a new window will be opened for the graph.
------------	--

Otherwise the graph will be displayed in a new tabbed sheet in a the currently selected graph window - if any.

OpenGroup

OpenGroup [/text] [/overwrite] [/spice3] [/spice2] [/purge] [/ign]
 [/forcereadopen] [/deleteonclose] *filename*]

Reads in a data file.

/text	If specified, data file is assumed to be in text format. Otherwise the file is input as a SIMetrix binary data file as saved by the SaveGroup command. See “Data Files Text Format” on page 447 for full details on the text format.
/overwrite	Forces existing group of the same name to be overwritten. If not specified, the group being read in will be renamed if a group of the same name already exists.
/spice3	If specified, <i>filename</i> , will be read in as a SPICE3 raw file. OpenGroup will readin the whole file into RAM. This may be inappropriate if the file is large. The command OpenRawFile (page 395) is usually a better choise for reading SPICE3 raw files as this rewrites the data to a native data file for access on demand.
/spice2	If specified, file will be read in as a SPICE2 raw file as generated by SPICE2g.6. This is an unsupported feature.
/purge	If specified, the loaded data group will be treated like a normal simulation group and will be automatically deleted after a specified number of runs. Otherwise it will not be deleted unless the user does so explicitly - e.g. by using the Graphs and Data>Delete Data Group... menu (which uses DelGroup).
/forcereadopen	If specified, read lock is ignored. The read lock prevents a data file from being opened for read and would typically be set when the file is being written out during a simulation. This switch overrides the lock.
/deleteonclose	If specified, the file will be marked as volatile and will be deleted once it is no longer needed.
<i>filename</i>	Name of file to be input. If not specified, an <i>open file dialog box</i> will be opened allowing the user to choose from available files.

OpenGroup creates a new Group. If /text is not specified then the name of the group will be that with which it was stored provided the name does not conflict with an existing group. If there is a conflict the name will be modified to be unique unless /overwrite is specified in which case the original group will be destroyed. If /text is specified then the group will be named *textn* where *n* is chosen to make the name unique.

OpenPrinter

OpenPrinter [/portrait] [/numcopies *num_copies*] [/index *index*] [/title *title*] [/printer *printer_name*]

Starts a print session. This may be used for customised or non-interactive printing. See “[Non-interactive and Customised Printing](#)” on page 463.

<i>/portrait</i>	If specified, print will be in portrait orientation, otherwise it will be landscape.
<i>num_copies</i>	Number of copies to print.
<i>printer_name</i>	Specify printer by name. If omitted, printer will be defined by its index (see below) or the application default printer will be used.
<i>index</i>	Printer to use. This can be found from the function <code>GetPrinterInfo</code> (page 185). If omitted, the application default printer will be used.
<i>title</i>	Title of <i>print job</i> . This is used to identify a print job and will be displayed in the list of current print jobs that can be viewed for each installed printer from control panel. <i>title</i> is not printed on the final document.

OpenRawFile

OpenRawFile [/purge] [/bufsize *buffer_size*] [/spice2] *rawfile* [*datafile*]

Opens a SPICE 3 format ASCII raw file.

<i>/purge</i>	If specified, the loaded data group will be treated like a normal simulation group and will be automatically deleted after three runs. Otherwise it will not be deleted unless the user does so explicitly - e.g. by using the Graphs and Data Delete Data Group... menu.
<i>/bufsize buffer_size</i>	Specifies the percentage proportion of installed RAM that is used for buffering the data. See Notes below for more details. Default value is 10 (%).
<i>/spice2</i>	If specified, <i>datafile</i> is assumed to be in SPICE format. This is an unsupported option
<i>/csdf</i>	If specified, the <i>datafile</i> is assumed to be in CSDF format
<i>rawfile</i>	Raw file to open.
<i>datafile</i>	SIMetrix data file to which data is written - see Notes. If omitted, a file will be created in the temporary data directory as specified by the <code>TempDataDir</code> option setting.

Notes

The command reads the raw file and writes the data out to a SIMetrix native data file. It then loads the SIMetrix native data file as if it were created by a SIMetrix

simulation. The SIMetrix data file format is more efficient than the raw file format as it stores the data for each vector in large contiguous blocks. The raw file format stores data on a per simulation point basis which leaves the data for multiple vectors interleaved. This arrangement makes data recovery for a single vector slow.

To perform the reformatting, the command needs to buffer the rawfile data in RAM while writing the data out to the SIMetrix data file. The amount of RAM space allowed for this controls the size of the contiguous blocks in the SIMetrix data file. The larger these blocks are, the faster the read in time for each vector. This is the same issue that affects the simulator and which is explained in the *Simulator Reference* manual in Chapter 2 under “Configuration Settings”. Here RAM used for this can be controlled by the `/bufsize` switch value. Note that the RAM is only needed while this command is being executed.

Note that the data file generated by this command can be reloaded at a later time using the `OpenGroup` command (or menu `File|Data|load...`). By specifying the *datafile* argument you can choose the name and location of this file which can be useful for archival purposes.

OpenSchem

OpenSchem [`/cd`] [`/readonly`] [`/backup`], *filename*

Reads a schematic file and draws it in a new schematic window. If the schematic is already open, it will be brought into view.

<code>/cd</code>	If specified, the directory holding <i>filename</i> is made current.
<code>/readonly</code>	Opens schematic in read-only mode. When opened in this mode, the file is not locked so that other users may open the file and write to it. If the file is already opened in non-readonly mode by another user, this switch must be specified in order to be able to open the file.
<code>/backup</code>	Restore temporary backup file. Same as normal restore except: The ‘modified’ flag is restored to its state when the file was saved. Normally the ‘modified’ flag is cleared. The pathname is restored to the path of the original file (if any) not the path of the backup file This command assumes that the original file was saved using <code>SaveSchem</code> with the <code>/backup</code> switch. This switch is used for the save/restore session feature and for recovering auto-saved schematics after an unexpected program exit.
<i>filename</i>	The schematic file to be input.

OpenSimplisStatusBox

OpenSimplisStatusBox

Opens the SIMPLIS simulation status box.

See Also

[“CloseSimplisStatusBox” on page 351](#)

OptionsDialog

OptionsDialog

Opens the options dialog box. This is the action performed by the menu File|Options|General.... All option processing is performed directly by this command.

Pan

Pan *x y*

Pans schematic.

<i>x</i>	Movement in x direction. A positive value moves the schematic to the left.
<i>y</i>	Movement in y direction. A positive value moves the schematic up.

Pause

Pause

Pauses current simulation (if any). Note that this command can only be executed by assigning it to a key or menu item with the direct execution option specified (option flag 5). For more information see [“User Defined Key and Menu Definitions” on page 434](#)

A paused simulation can be restarted with the Resume command ([page 410](#)).

PinDef

PinDef *name number left|top|right|bottom [vis|novis]*

This is a symbol definition command. Creates a pin for a generated symbol. PinDef must not be used with AddPin ([page 343](#)) within the same symbol definition.

<i>name</i>	Pin name. May not contain spaces
<i>number</i>	Pin number. Pin numbers must begin at 1 and be contiguous. Determines the order in which the pins appear on the device's netlist entry.
<i>left top right bottom</i>	Specifies whether pin should go on left, right, top or bottom of symbol. Exact location is decided by SIMetrix but pins are placed in order of definition.
<i>vis novis</i>	Specify if pin name is to be visible on symbol.

PlaceCursor

PlaceCursor *[/main x_main y_main] [/datum x_datum y_datum]*

Positions graph cursors if they are enabled.

/main x_main y_main

Location of main measurement cursor. Position is determined by *x_main*. *y_main* is only used for non-monotonic curves (e.g. Nyquist plots) where there is more than one y value for a given x value.

/datum x_datum y_datum

Location of reference cursor. Position is determined by *x_datum*. *y_datum* is only used for non-monotonic curves (e.g. nyquist plots) where there is more than one y value for a given x value.

Plot

Plot *[/autoXlog]*
[/autoYlog]
[/autoAxis]
[/axisId axis_id]
[/coll]
[/dig]
[/loglog]
[/name curve_name]
[/newAxis]
[/newGrid]
[/newSheet]
[/new]
[/select]
[/title title]
[/xauto]
[/xdelta x_grid_spacing]
[/xl x_low_limit x_high_limit]
[/xlabel x_label_name]
[/xlog]
[/xunit x_unit_name]
[/yauto]
[/ydelta y_grid_spacing]
[/yl y_low_limit y_high_limit]
[/ylabel y_label_name]
[/ylog]
[/yunit y_unit_name]

[y_expression]

[*x_expression*]

Plot can be used to add a new curve to an existing graph created with Plot or to change the way it is displayed.

/autoXlog	If specified, the x-axis will be logarithmic if the x-values are logarithmically spaced.
/autoYlog	Same as /autolog except that if x-values are logarithmically spaced, the Y axis will be logarithmic
/autoAxis	Does nothing. For compatibility with Curve command.
/axisId <i>axis_id</i>	Does nothing. For compatibility with Curve command.
/coll	Does nothing. For compatibility with version 3.1 and earlier.
/dig	If specified, the curve will be plotted on a digital axis. Digital axes are stacked on top of main axes and are sized and labelled appropriately for digital waveforms.
/loglog	Forces both y and x axes to be logarithmic
/name <i>curve_name</i>	If specified, curve will be named <i>curve_name</i> .
/newAxis	Does nothing. For compatibility with Curve command.
/newGrid	Does nothing. For compatibility with Curve command.
/newSheet	Creates a new empty sheet. Does not plot any curves.
/new	Opens a new graph window.
/select	If specified, the new curve will be selected.
/title	Specify title of graph. Followed by ...
<i>graph_title</i>	Graph title
/xauto	Does nothing. Included for compatibility with Curve command.
/xdelta	Specify spacing between major grid lines on x-axis. Followed by...
<i>x_grid_spacing</i>	Real. For default spacing use '0'.
/xl	Use fixed limit for x-axis. Followed by ...
<i>x_low_limit</i>	Real. Lower limit of x-axis.
<i>x_high_limit</i>	Real. Higher limit of x-axis.
/xlabel	Specify label for x-axis. Followed by...
<i>x_label_name</i>	Text String. Label name. If it contains spaces, whole string must be enclosed in quotes ("").
/xlog	Only effective when graph sheet is empty. Forces logarithmic x-axis.
/xunit	Specify units for x-axis (Volts, Watts etc.). Followed by ...
<i>x_unit_name</i>	Text string. Unit name. If it contains spaces, the whole string

	must be enclosed in quotes (""). You should not include an engineering prefix (m, K etc.).
<code>/yauto</code>	Does nothing. Included for compatibility with Curve command.
<code>/ydelta</code>	Specify spacing between major grid lines on y-axis. Followed by...
<code>y_grid_spacing</code>	Real. For default spacing use '0'.
<code>/yl</code>	Use fixed limit for y-axis. Followed by ...
<code>y_low_limit</code>	Real. Lower limit of y-axis.
<code>y_high_limit</code>	Real. Higher limit of y-axis.
<code>/ylabel</code>	Specify label for y-axis. Followed by...
<code>y_label_name</code>	Text String. Label name. If it contains spaces, whole string must be enclosed in quotes ("").
<code>/yunit</code>	Specify units for y-axis. Followed by ...
<code>y_unit_name</code>	Text string. Unit name. Other comments as for x unit name.
<code>y_expression</code>	Text string. Expression describing curve to be added to graph.
<code>x_expression</code>	Text string. Expression describing x values of curve defined by y expression. If omitted, reference of <code>y_expression</code> will be used.

`/autoxlog` and `/autoxylog` log test

The x-values are deemed to be logarithmically spaced if three values satisfy the following:

$$1.0000001 > x1*x1/(x0*x2) > 0.9999999$$

Where:

$x0 = x[0]$ i.e the first point in the data

If there are an even number of points:

$x1 = x[\text{len}/2-1]$ where len is the number of points in the data

$x2 = x[\text{len}-2]$

If there are an odd number of points

$x1 = x[(\text{len}-1)/2]$ where len is the number of points in the data

$x2 = x[\text{len}-1]$

If there are fewer than three points or any of the values is less than or equal to zero, a linear axis will be selected.

PreProcessNetlist

```
PreProcessNetlist [/inAppend extraInputLines]  
[/simulator SIMPLIS|SIMetrix] inFile outFile
```

Pre-processes the specified netlist. The netlist pre-processor was developed for use with the SIMPLIS simulator but is general purpose in nature and may also be used with

SIMetrix. Currently this command is automatically called when a SIMPLIS simulation is run from the GUI. It is not currently used for SIMetrix simulations.

<i>inFile</i>	Input file name to be processed
<i>outFile</i>	File to receive result
<i>extraInputLines</i>	Additional lines appended to input file. Each line separated by a semi-colon ';'

The `/simulator` switch allows the specification of the simulator that the netlist is intended for and affects library searching and the effect of the `.SIMULATOR` control. The default value is `SIMPLIS`.

Documentation for the pre-processor language syntax may be found in the *SIMPLIS Reference Manual*. This is supplied in hardcopy form with SIMetrix/SIMPLIS but is also available as a PDF file on the install CD and at the SIMetrix Technologies Ltd. web site at www.simetrix.co.uk.

PrintGraph

```
PrintGraph [ /margin left top right bottom ] [ /major on|off ] [ /minor
on|off ] [ /caption caption ] [ /mono ] [ dim_left dim_top dim_right
dim_bottom ]
```

Prints the current graph sheet.

left top right bottom Page margins in mm.

`/major on|off` Specify whether major grid lines should be printed. Default is on.

`/minor on|off` Specify whether minor grid lines should be printed. Default is on.

caption Caption printed at the bottom of the page.

dim_left, dim_top, dim_right, dim_bottom

Dimensions and position of printed image on page. Values are relative to page size less the specified margins in units equal to 1/1000 of the page width/height. The default is 0 0 1000 1000 which would place the image to fill the entire area within the margins. 0 500 1000 1000 would place the image at the bottom half of the page. 0 0 2000 1000 would place the left half of the image in the full page while -1000 0 1000 1000 would place the right half. This allows the printing on multiple sheets. Note that if values greater than 1000 or less than 0 are used, part of the printed image will lie in the margins. This provides a convenient overlap for multiple sheets.

`/mono` If specified, the graph will be printed in black and white

PrintSchematic

```
PrintSchematic [ /caption caption ] [ /fixed grid_size ] [ /margin left top
```

Script Reference Manual

right bottom [/mono] [*dim_left dim_top dim_right dim_bottom*]

Prints the current schematic.

left top right bottom Page margins in mm.

caption Caption printed at the bottom of the page.

grid_size If specified, fixed scaling will be used. *grid_size* is the size of a single grid square on the printed sheet in inches. Otherwise the schematic scale will be chosen to fill the print area. The scaling is *isotropic*. That is the aspect ratio will be maintained.

dim_left, dim_top, dim_right, dim_bottom

Dimensions and position of printed image on page. Values are relative to page size less the specified margins in units equal to 1/1000 of the page width/height. The default is 0 0 1000 1000 which would place the image to fill the entire area within the margins. 0 500 1000 1000 would place the image at the bottom half of the page. 0 0 2000 1000 would place the left half of the image in the full page while -1000 0 1000 1000 would place the right half. This allows the printing on multiple sheets. Note that if values greater than 1000 or less than 0 are used, part of the printed image will lie in the margins. This provides a convenient overlap for multiple sheets.

/mono If specified, the graph will be printed in black and white

Probe

Probe [/type 1|2|P|N]

Moves mouse cursor to currently selected schematic, changes cursor shape to a symbol depicting an oscilloscope probe then suspends command execution. When any mouse key is clicked, the cursor shape reverts to normal and command execution is resumed. Probe does not suspend commands executed directly on assignment to keystrokes or menu items. This allows the Cancel command, when assigned to a key or menu, to terminate a probe command. Note that the Probe command completes on both up and down strokes of a mouse key.

/type 1|2|P|N Alters slightly the cursor shape by adding a single character as follows:

1 - adds '1'
2 - adds '2'
P - adds a '+' character
N - adds a '-' character

Prop

Prop [/show | /hide | /noadd] [/flags *value*|*property*] [/toggle]
[/showName] [/hideName] [/hideNew] [/pinloc *pinnumber*]
property_name property_value

Modifies a property value of a schematic component if it exists. If it doesn't exist the property is added.

<i>property_name</i>	Which property to modify.
<i>property_value</i>	New value.
/show	Make property visible
/hide	Make property invisible
/flags <i>value</i>	If specified, changes/assigns the <i>flags</i> value of the property. The flags value defines the properties attributes. How this number is composed is detailed below.
/flags <i>property</i>	If specified, copies the flags value from the specified property so the new/changed property defined by <i>property_name</i> will have the same flags as the already existing <i>property</i> . The <i>flags</i> define the property's attributes.
/noadd	If specified, property will not be added if the instance does not already possess it.
/toggle	Toggles hide/show state. I.e. if property is already hidden it will be made visible and vice-versa.
/showName	If specified, the name of the property will be made visible along with its value in the form "name=value".
/hideName	If specified, the name of the property will be hidden.
/hideNew	Hide property value if a new property is being added. If the property already exists, its visibility will remain unchanged.
/pinloc <i>pinnumber</i>	If specified, the property will be positioned at a fixed location next to the pin specified by <i>pinnumber</i> .

Attribute flags

The attributes flag value is a 16 bit number with each bit having a defined function. These bits are defined in the following diagram

Bit 0,1	Auto text location for normal orientation: 00 Left 01 Top 10 Right 11 Bottom
	If fixed position, value controls left-right justification: 00 left 01 centre 10 right
	Unused set to 0
Bit 3,4	Auto text location for 90 degree rotated orientation: 00 Left 01 Top 10 Right 11 Bottom
	If fixed position, value controls top-bottom justification: 00 top 01 baseline (means the base for upper case characters. The tails of some lower case characters go below the baseline.)
Bit 5	Unused set to 0
Bit 6	Visibility 0 Visible 1 Hidden
Bit 7	Protected status 0 Not protected 1 Protected
Bit 8	Location method 0 Auto (use bits 0,1,3,4 to define) 1 Fixed pos (actual location can only be defined in symbol)
Bit 9	Text scale method 0 Optimum readability 1 Linear
Bit 10	Does property text define select border 0 No 1 Yes

Bits 11-13	Font index. 0 Default 1 Caption 2 Free text 3 Annotation 4 User 1 5 User 2 6 User 3 7 User 4
Bit 14	rotated. Property at 90 degrees to symbol orientation. Ignored if location method = auto.
Bit 15	Display property name with value.
Bit 16	Resolve symbolic value if specified. Currently only three are permitted namely, <version>, <date> and <time>. If this flag is set any of the above strings are found in the property, they will be replaced by their value. <version> will be replaced by an integer that is incremented each time the schematic is saved. <date> and <time> will be replaced by the date and time of the schematic file respectively.

The final value has to be entered as a decimal value. Note that attributes are usually edited using the popup menu Edit Properties... dialog.

Examples

To change a R3's component reference to R4 (i.e. change its *ref* property from R3 to R4) select R3 then enter:

```
Prop ref R4
```

Protect

Protect

Protects selected schematic components. Protected components cannot be selected. This command is used for schematic worksheets so that they remain in a fixed position. The Unprotect command removes protected status.

Quit

Quit

Terminates SIMetrix. If there are any modified schematics open, the user will be prompted to save them first.

RD

Rd *directory_name*

Remove a directory. Rd is similar to the DOS RD and RMDIR commands.

directory_name Name of directory to be removed.

ReadLogicCompatibility

ReadLogicCompatibility *filename*

Reads a file to define the compatibility relationship between logic families. For an example of a compatibility table, see the file COMPAT.TXT which you will find in the SCRIPT directory. This file is actually identical to the built-in definitions except for the "UNIV" family which cannot be redefined.

Please refer to the "Digital Simulation" chapter of the *Simulator Reference Manual* for full details on logic compatibility tables.

File format

The file format consists of the following sections:

Header

In-Out resolution table

In-In resolution table

Out-Out resolution table

Header:

The names of all the logic families listed in one line. The names must not use the underscore ('_') character.

In-Out resolution table:

A table with the number of rows and columns equal to the number of logic families listed in the header. The columns represent outputs and the rows inputs. The entry in the table specifies the compatibility between the output and the input when connected to each other. The entry may be one of three values:

OK	Fully compatible
WARN	Not compatible but would usually function. Warn user but allow simulation to continue.
ERR	Not compatible and would never function. Abort simulation.

In-In resolution table

A table with the number of rows and columns equal to the number of logic families listed in the header. Both column and rows represent inputs. The table defines how inputs from different families are treated when they are connected. The entry may be one of four values:

ROW	Row take precedence
COL	Column takes precedence
OK	Doesn't matter. (Currently identical to ROW)
ERR	Incompatible, inputs cannot be connected.

Out-out resolution table

A table with the number of rows and columns equal to the number of logic families listed in the header. Both column and rows represent outputs. The table defines how outputs from different families are treated when they are connected. The entry may be one of four values:

ROW	Row take precedence
COL	Column takes precedence
OK	Doesn't matter. (Currently identical to ROW)
ERR	Incompatible, outputs cannot be connected.

RebuildSymbols

RebuildSymbols

The installed symbol library is usually stored in RAM during normal operation. When a symbol is needed, the modified date of original source file is checked and if it has changed, that library file will be reloaded. This happens anyway whenever a symbol is required for any purpose.

RebuildSymbols forces the checking of all stored symbol libraries and any that are out of date will be reloaded from the source file.

There aren't many reasons for using this command. However, it is sometimes useful to call it in the startup script so that the symbols are automatically loaded when the program starts. Normally the symbols aren't loaded until they are first needed and this can introduce a slight delay.

Redirect

Redirect /err | /out [*filename*]

Redirects messages (i.e. text which is normally displayed in the message window) to a file. See also RedirectMessages below.

filename Name of file to which messages are sent. If not specified messages are sent to the message window.

One or both of /err or /out must be specified:

/err Specifies that error and warning messages are to be redirected.

/out Specifies that messages other than errors and warnings are to be redirected.

RedirectMessages

RedirectMessages on *filename* | dup *filename* | off | flush

Redirects all command shell messages to a file. Unlike Redirect (above) this directs *all* messages to the file. Redirect doesn't redirect everything.

filename File to receive redirected output

on	Switch on redirect. All messages will go to <i>filename</i> and no output will appear in the message window
dup	Switch in redirect. All messages will go to <i>filename</i> <u>and</u> to the message window
off	Switch off redirect
flush	Flush file. When redirect is switched on, messages are buffered before being written to the target file. This will flush the buffer so that the file contents will be up to date.

Redo

Redo

Reverses the most recent undo operation.

RegisterUserFunction

RegisterUserFunction *Function-Name Script-Name* [*min-number-args*]
[*max-number-args*]

Creates a user defined function based on a script.

<i>Function-Name</i>	Name of function. This must start with a letter and contain only letters, digits and underscores. The name must not be one of the built-in functions.
<i>Script-Name</i>	Name of script that will be called to execute function.
<i>min-number-args</i>	Minimum number of arguments required by the function. Range 0 - 7. Default=0
<i>max-number-args</i>	Maximum number of arguments that may be supplied to the function. Range 0 - 7. Default=7

Notes

When an expression is evaluated that calls the function defined by this command, the specified script will be called. The script receives the arguments to the function through its argument numbers 2-8. (There is a maximum limit of seven arguments). The function's returned value is the script's first argument passed by reference.

Further details including an example are given in [“User Defined Script Based Functions” on page 461](#).

RenameLibs

RenameLibs [/report] [/check] [/log *logfile*] *filename suffix* [*catalog-file user-catalog-file*]

Runs the rename model utility. This renames models inside installed model files if they are found to have duplicates. This command is called by the `rename_libs` script which is documented in the User's Manual.

<i>filename</i>	Name of model library file or file spec to be processed. This may include '*' or '?' wild card characters. Any models within this file that have duplicates already installed in the global model library will be renamed using the suffix supplied.
<i>suffix</i>	Suffix applied to duplicate model name.
<i>catalog-file</i>	Usually called OUT.CAT. If specified alongside <i>user-catalog-file</i> , any user association of renamed models will be appropriately modified.
<i>user-catalog-file</i>	Usually called USER.CAT. See <i>catalog-file</i> above.
<i>/report</i>	If specified a report of progress will be displayed in the command shell.
<i>/check</i>	If specified a dummy renaming process will be performed. All reports, logs and messages will be output but no actual renaming will take place
<i>logfile</i>	If specified, all renamed models will be listed in <i>logfile</i> .

RepeatLastMenu

RepeatLastMenu *menuname top-menu-name*

Executes the menu most recently selected by the user. SIMetrix remembers the last command executed for each top level menu and this menu must be specified with this command.

<i>menuname</i>	Identifies the window type that owns the menu. See DefMenu (page 368) command for list of possible values.
<i>top-menu-name</i>	The top level menu name. For a fixed menu, this is the name that appears in the menu bar. For popup menus, the name "\$popup\$" must be supplied.

Reset

Reset

Frees memory associated with most recent simulation run.

It is not normally necessary to use this command unless available memory is low and is needed for plotting graphs or other applications. Note that Reset does not delete the data generated by a simulation only the internal data structures set up to perform a run. These are automatically deleted at the beginning of a new run.

RestartTran

RestartTran *new-tran-stop-time*

Restarts a transient simulation that had previously run to completion. To work, the most recent simulation must have been a transient analysis. If another analysis has

since been run or if the analysis has been cleared using the Reset command, this command will be inoperative.

new-tran-stop-time The restarted run will continue until it reaches this time.

RestDesk

RestDesk

Restores all windows to positions saved with last SaveDesk command.

Resume

Resume

Resumes a previously paused simulation.

RotInst

RotInst [*orientation*]

Changes orientation of selected items.

<i>orientation</i>	Integer from 0 to 7 to specify how symbol should be oriented:
0	No change
1	Rotate clockwise 90°
2	Rotate clockwise 180°
3	Rotate clockwise 270°
4	Mirror through vertical axis
5	Mirrored + 90° Rotation
6	Mirrored + 180° Rotation
7	Mirrored + 270° Rotation

Run

Run [/check] [/an *analysis_spec*] [/force] [/file] [/nofile] [/options *optionsString*] [/list listFile] [/nolist] [/nodata] [/collection *collection_name*] [/sweep start|continue|finish] [/append *groupname*] [/label *div_label*] *netlist_name* [*data_filename*]

Runs a simulation on specified netlist.

/check	If specified, simulation is not run but netlist is read in and all checks are performed.
/an <i>analysis_spec</i>	If specified, any analysis controls (e.g .TRAN, .AC etc.) in the netlist are ignored and the control in <i>analysis_spec</i> is executed instead.
/force	<i>data_filename</i> will be overwritten if it already

Chapter 6 Command Reference

	exists without prompting user. Otherwise a dialog box will be opened allowing user to select a new file if required.
<code>/file</code>	Does nothing. In earlier versions (pre 3.1) this had to be specified to force simulation data to be output to a file. This is now the default behaviour. Specify <code>/nofile</code> to force data to be stored in RAM.
<code>/nofile</code>	If specified, simulation data is stored in RAM.
<code>/options <i>optionsString</i></code>	Simulator options settings. <i>optionsString</i> may be anything that can be placed after a <code>.OPTIONS</code> control. (Must be enclosed in double quotation marks if <i>optionsString</i> contains spaces).
<code>/list <i>listFile</i></code>	Overrides default name for list file.
<code>/nolist</code>	Inhibits creation of list file.
<code>/nodata</code>	Only data explicitly specified by <code>.PRINT</code> or <code>.KEEP</code> controls will be output. Usually all top level data is saved. Equivalent to placing <code>“.KEEP /nov /noi /nodig”</code> in netlist.
<code>/collection <i>collection_name</i></code>	Attaches group generated by run to collection of name <i>collection_name</i> . This is now obsolete and may not be supported in future versions. Collections were used in earlier versions to group data created by multiple runs such as Monte Carlo and stepped analyses. These runs now create a single group containing <i>multi-division</i> vectors. This is explained on page 38 .
<code>/sweep</code>	May be set to ‘start’, ‘continue’ or ‘finish’. This is used to create linked runs that save their data to the same group using multi-division vectors. The first run in such a sequence should specify <code>‘/sweep start’</code> while the final run should specify <code>‘/sweep finish’</code> . All intermediate runs should specify <code>‘/sweep continue’</code> . All runs except the first must also specify <code>‘/append’</code> - see below.
<code>/append <i>groupname</i></code>	Append data created to specified group name which would always be the data group created by the first run in the sequence. <code>‘/sweep continue’</code> or <code>/sweep finish’</code> must also be specified for this to function. The data is appended by adding new divisions to existing vectors so creating or extending a multi-division vector.
<code>/label <i>division_label</i></code>	Used with <code>/sweep</code> to name the division of a linked run.

<i>netlist_name</i>	Input netlist filename.
<i>data_filename</i>	Specifies path name of file to receive simulation data. If omitted, the data is placed in a temporary data file.

Notes

The Run command does not run a simulation on the currently open schematic but on the specified netlist. Normally a run is initiated using the Simulator|Run menu item. This annotates the schematic then generates the netlist using the Netlist command. Run is then executed specifying the new netlist.

The Run command may also be used to run a simulation on a netlist generated by hand or by another schematic editor.

Linking Runs

The data from multiple runs may be linked together in the same manner as multi-step runs such as Monte Carlo. This makes it possible to develop customised multi-step runs using the script language. Simple multi-step runs may be defined using the simulator's built in features which cover a wide range of applications. The simulator's multi-step features allow the stepping of a single component or a parameter which can define several components. But it doesn't allow, for example, a complete model to be changed, or any kind of topological changes.

The script language may be used to control multiple runs of a circuit with no limit as to the changes that may be performed between each run. In such situations it is useful to be able to organise the data in the same way that the native multi-step facilities use. This can be done by linking runs using the /sweep, /append and /label switches. By running simulations in this manner, the data generated by the simulator will be organised using multi-division vectors which are similar to 2 dimensional arrays. See [page 38](#) for more details.

Care must be taken when making topological changes between runs. Names of nodes that are of interest must always be preserved otherwise the data generated for their voltage may be lost or mixed up with other nodes. Note also that the data for new nodes created since the first run will not be available. The same problems arise for device pin currents.

Note that the netlist for a linked run must specify a single analysis only. E.g. a single .TRAN or .AC but not both. Also, do not add .OP lines to the netlist.

Linked Run Example

```
** First run
Run /sweep start /label "Run=1" netlist.net
** save group name
Let grpl = (Groups()) [0]

... changes to netlist
** second run
Run /sweep continue /label "Run=2" /append {grpl} netlist.net

... changes to netlist
```

```

** third run
Run /sweep continue /label "Run=3" /append {grp1} netlist.net

... changes to netlist
** fourth and final run
Run /sweep finish /label "Run=4" /append {grp1} netlist.net

```

RunSIMPLIS

RunSIMPLIS [/fresh] [/append] [/label *division_label*] [/sweep start|continue|finish] [/checkAbort] *filename*

Runs the SIMPLIS simulator. Note that you must have a SIMetrix/SIMPLIS license for this command to work.

<i>/fresh</i>	Instructs SIMPLIS to run simulation afresh and not to use any state information saved from previous runs.
<i>/sweep</i>	Used for multi-step runs. See Run command above for details
<i>/append</i>	Append data to current group. Otherwise creates a new data group
<i>division_label</i>	Used with <i>/sweep</i> to name the division of a linked run
<i>/checkAbort</i>	Instructs SIMPLIS to check abort requests.
<i>filename</i>	Name of file containing the SIMPLIS netlist. If a full path is not supplied, <i>filename</i> will be assumed to be relative to the current directory. Note that the extension of the file must always be supplied; no default is assumed.

The RunSIMPLIS command will not pre-process the netlist. This must be done separately using the PreProcessNetlist ([page 400](#)) command.

Notes

RunSIMPLIS is the primitive SIMetrix command that launches SIMPLIS. However, when running a simulation on a schematic, a number of other activities are performed. These include pre-processing the netlist generated by the schematic editor and also resolving a trigger device for POP analysis. If you wish to simulate a schematic in exactly the same manner as the Run menu, you need to execute the script `simplis_run`. This simulates the currently open schematic. The full source for `simplis_run` can be found on the install CD.

Save

Save [/all] [/interactive] [/v25]

Saves the currently selected schematic.

<i>/all</i>	If specified, all open schematics will be saved.
<i>/interactive</i>	If specified the user will be prompted to supply the path name for unnamed schematics. Otherwise, unnamed schematics will not be saved and an error message will be displayed.

/v25 If specified, the schematic will be saved in format version 2.5. This is compatible with SIMetrix versions 2.5, 3.0, 3.1 and 4.0. Otherwise the schematic will be saved using a format that may be read by all versions from 4.1.

SaveAs

SaveAs [*/force*] [*/v25*] [*/ascii*] [*/writeSymbol*] [*/ui user_index*] [*/tab tab_id*] *filename*

Saves the currently selected schematic.

<i>filename</i>	Name of file to which schematic is saved. (<i>filename</i> is not optional as it was with earlier versions of SIMetrix.)
<i>/force</i>	If specified and <i>filename</i> already exists it will be overwritten without prompting. Otherwise user will be given the option to cancel the operation, specify a new file or overwrite the existing file.
<i>/v25</i>	If specified, the schematic will be saved in format version 2.5. This is compatible with all SIMetrix versions 2.5 and later. Otherwise the schematic will be saved using format version 4.1 which is compatible with versions 4.1 and later
<i>/ascii</i>	Forces the file to be written in ASCII format. Otherwise the format of the existing file is used. If the file doesn't exist then it will be saved in binary format
<i>/writeSymbol</i>	If the schematic being saved has an embedded symbol (that forms part of a hierarchical component), the symbol will be written out if this switch is specified. Otherwise the symbol will not be written out. If <i>filename</i> already exists and already has a symbol, that symbol will remain intact if this switch is <i>not</i> specified.
<i>user_index</i>	User index as returned by the function GetWindowNames (page 208). Specifies which window to save. If omitted, the currently selected window is saved
<i>tab_id</i>	Tab id - used to specify which tabbed sheet within a schematic window is to be saved. <i>tab_id</i> is a number between zero and 1 less than the number of tabbed sheets in the window. The function GetOpenSchematics (page 183) can be used to determine the number of tabs open in a window.

SaveDesk

SaveDesk

Saves the current window positions. They can be restored using RestDesk. (or menu File|Windows|Restore Desktop)

SaveGraph

SaveGraph [/version *data_version*] [/id *graph_id*] *filename*

Saves the currently selected graph to a binary file. This can subsequently be restored using OpenGraph ()

<i>data_version</i>	Data version to use. Set to 4.0 to save in a format that is compatible with version 5.4 and earlier SIMetrix versions. Otherwise version will be 6.0 which requires SIMetrix version 5.5 or later. The latter version uses 64bit pointers and thus allows curves using more than 4GBytes of data. (Requires a 64 bit version and OS)
<i>graph_id</i>	Graph object id. If more than one graph is displayed, <i>graph_id</i> can be used to identify which graph is saved. If omitted the currently selected graph is used. All currently open graphs can be obtained using GetGraphObjects('graph') , while GetGraphTabs() can be used to obtain the graph objects within a single window.
<i>filename</i>	Path of file.

SaveGroup

SaveGroup [/force] [/version *version*] [*filename*]

Saves the current group in binary format

<i>/force</i>	If is specified, any existing file will be overwritten without prompting
<i>version</i>	Can be 0, 4.0 4.5 or 6.0. 0 means use default which will be the latest version. 4.0 will be useable with SIMetrix versions 4.0 and later. 4.5 requires SIMetrix version 4.5c or later while 6.0 requires version 5.5 or later
<i>filename</i>	Save to the <i>filename</i> . Data can later be restored with the OpenGroup command. If <i>filename</i> is not specified a dialog box will be opened allowing the user to choose from available files.

SaveRhs

SaveRhs [/nodeset] *filename*

Creates a file containing every node voltage, inductor current and voltage source current calculated at the most recent analysis point. The values generated can be read back in as nodesets to initialise the dc operating point solution. There are a number of applications for this command - see below.

<i>/nodeset</i>	If specified the values are output in the form of a .nodeset command which can be read back in directly. Only node voltages are output if this switch is specified. Otherwise, currents in voltage sources and inductors are also output.
-----------------	---

filename File where output is written

This command is intended as an aid to DC operating point convergence. Sometimes the dc operating point solution is known from a previous run but took a long time to calculate. By applying the known solution voltages as nodesets prior to the operating point solution, the new DC bias point will be found much more rapidly. The method is tolerant of minor changes to the circuit. The old solution may not be exact, but if it is close this may be sufficient for the new solution to be found quickly.

If SaveRhs is executed after an AC analysis, the values output will be the real part only.

SaveSnapshot

SaveSnapshot

Saves the current state of a transient analysis to a snapshot file. This can be retrieved later to initialise an AC analysis. For more information on snapshots see the *User's Manual* and the *Simulator Reference Manual*.

SaveSymbol

```
SaveSymbol [ /force ] [ /local ] [ /lib lib_symbol_name ] [ /comp ] [ /file  
pathname ] [ /flags flags ] [ /ascii ] symbol_name [ symbol_description [  
catalog ] ]
```

Save a symbol to a library or as a component. Source may be the current symbol editor symbol or a specified symbol in the global library.

<i>/force</i>	No longer active and supported for backward compatibility only.
<i>/local</i>	No longer active and supported for backward compatibility only. Originally instructed to save symbol to current schematic only. This now happens anyway if neither <i>/file</i> nor <i>/comp</i> are specified.
<i>/lib lib_symbol_name</i>	Use specified library symbol as source instead of the symbol editor. <i>lib_symbol_name</i> must be the internal name for a symbol in an installed library.
<i>/comp</i>	Saves symbol as a component to path <i>symbol_name</i> .
<i>/file pathname</i>	Symbol saved to specified library file. This is ignored if <i>/comp</i> is specified. If a full path is not supplied, the path will be relative to the SymbolLibs directory.
<i>/flags flags</i>	If flags=1, symbol is saved with tracking enabled. This forces all instances of the symbol to always be loaded from the global symbol library rather than from the local schematic. This is the action of the All references to symbol automatically updated check box in the symbol editor's File Save... dialog box.
<i>/ascii</i>	Only effective if <i>/comp</i> specified. Symbol is saved to component file using the ASCII format.

<i>symbol_name</i>	Name of symbol. This is known as the ‘internal name’ in the user interface. This is the name that the software uses to identify the symbol. It is stored in schematic files and it is used for a number of script functions and commands, for example the Inst (page 383) command to place a symbol uses this name. This name may not contain spaces or special characters and cannot be changed once the symbol is created.
<i>symbol_description</i>	Symbol description. This is the name that is displayed in the dialog opened with Place From Symbol Library.... Unlike the <i>symbol_name</i> (above) it has no naming restrictions and can be changed at any time without affecting any existing instances of the symbol.
<i>catalog</i>	Symbol catalog. This determines how the symbol is categorised in Place From Symbol Library.... This may be a list of strings separated by semi-colons, each identifying a node in the tree list display shown in the place symbol dialog box.

SaveSymLib

SaveSymLib [*/v25*] [*/append*] [*/force*] [*/lib libname*][*/all*] [*/ascii*]
filename

Writes the entire global symbol library or a specified installed symbol library file to *filename*. Note that the action of this command has changed significantly from that of version 4.0 and earlier.

<i>/v25</i>	If specified, the file will be written in a format compatible with all SIMetrix versions 2.5, 3.0, 3.1 and 4.0. Otherwise the format used will work only with versions 4.1 or later
<i>/append</i>	Symbols are appended to <i>filename</i> . Otherwise <i>filename</i> will be overwritten if it already exists. Note that any symbols written that are already present in <i>filename</i> will be overwritten. It is not possible to have duplicate symbols within the same library file.
<i>/force</i>	Allows symbols to be written to an existing library file. Otherwise if <i>filename</i> is an existing installed library, the command will abort with an error message.
<i>/lib libname</i>	Name of library file to write out. This must be an installed file.
<i>/all</i>	Write out all installed symbols.
<i>/ascii</i>	Force ASCII format. If <i>not</i> specified the symbol library will be saved in the same format as the existing file or binary if it doesn't exist.
<i>filename</i>	File to receive symbols

ScriptAbort

ScriptAbort

Aborts execution of script. Note that this command can only be usefully executed from a key or menu item which has been defined with the direct execution option specified (option flag 5 or /immediate switch for DefMenu). See “[User Defined Key and Menu Definitions](#)” on page 434.

See Also

ScriptStep [page 418](#)
ScriptResume [page 418](#)
ScriptPause [page 418](#)

ScriptPause

ScriptPause

Pauses a script. Execution can later be resumed with ScriptResume or single stepped with ScriptStep. Note that this command is often executed from a key or menu item which has been defined with the direct execution option specified (option flag 5 or /immediate for DefMenu). ScriptPause is assigned to shift-F2 by default. Note that it is not possible to use the normal user interface while a script is paused. The main use of script pause is to allow single-stepping for debug purposes.

Scripts can be single stepped by executing ScriptPause immediately before starting the script. If the EchoOn option is also enabled, each line of the script as it is executed will be displayed in the message window. See “[Debugging Scripts](#)” on page 46.

See Also

ScriptStep [page 418](#)
ScriptResume [page 418](#)
ScriptAbort [page 417](#)

ScriptResume

ScriptResume

Resumes script that has been paused with ScriptPause.

See Also

ScriptStep [page 418](#)
ScriptPause [page 418](#)
ScriptAbort [page 417](#)

ScriptStep

ScriptStep

Steps a paused script by one command. See “[Debugging Scripts](#)” on page 46.

See Also

ScriptPause [page 418](#)

ScriptAbort [page 417](#)
 ScriptResume [page 418](#)

Select

Select [/wires] [[/prop *name* [*value*]] | [/wire *wirehandle*]] [/all]

Select items on selected schematic. If the /prop switch is not specified the interactive select mode is entered.

/wires	If specified, only wires will be selected. Otherwise both components and wires will be selected.
/prop <i>name value</i>	If <i>value</i> is specified, all components on the current schematic with property of name <i>name</i> and value <i>value</i> will be selected. If <i>value</i> is not specified then all components possessing the property <i>name</i> will be selected.
/wire <i>wirehandle</i>	Select wire with handle defined by <i>wirehandle</i> .
/all	If specified, all items on the current schematic sheet will be selected.

Notes

The /prop switch makes it possible to automate modification of component values using a script. For example, supposing you have a circuit with a resistor R2 and capacitors C4 and C5, you could modify the values of all of them with a script something like:

```
Unselect
Select /prop ref R2
Prop value 1.1k
Unselect
Select /prop ref C4
Prop value 120p
Unselect
Select /prop ref C5
Prop value 1.2n
```

The above script would change R2, C4 and C5 to 1.1k, 120p and 1.2n respectively.

SelectCurve

SelectCurve [/unselect] *curve_id*

OR

SelectCurve [/unselect] /all

Selects/unselects the identified curve or all curves

FORM1

Specified curve is selected or unselected.

curve_id Curve Id. Curve id is returned by the functions GetSelectedCurves ([page 187](#)), GetAxisCurves ([page 146](#)) and GetAllCurves ([page 143](#))

FORM2

All curves on currently selected graphs are selected or unselected.

BOTH CASES:

/unselect Curve or curves to be unselected.

SelectGraph

SelectGraph *id*

Switches the graph tabbed sheet to the graph specified by *id*.

id Graph id. See “[Graph Object Identifiers - the “ID”](#)” on [page 449](#) for more information

SelectLegends

SelectLegends [/unselect]

Selects or unselects all graph window legends.

/unselect If specified, all legends are unselected. Otherwise they are selected.

SelectSimulator

SelectSimulator *simulator-name*

Selects current simulator for selected schematic.

simulator-name Name of simulator to be selected. Current valid values are “SIMetrix” and “SIMPLIS”.

Set

Set [/temp] [*option_spec* [*option_spec...*]]

Defines an option.

/temp If specified, the setting will only remain for the duration of the current script execution. Value will return to its original setting when control returns to the command line.

option_spec Can be one of two forms:
Form1: *option_name*
Form2: *option_name* = *option_value*

option_name can be any of the names listed in the options section of the “Sundry Topics Chapter” of the *User's Manual*. For options of type Boolean, use form 1. For others, use form 2.

See Also

UnSet [page 430](#)

SetCurveName

SetCurveName *curve_id curve_label*

Changes curves label. This is the text displayed in the legend panel.

curve_id Curve Id. Curve id is returned by the functions GetSelectedCurves ([page 187](#)), GetAxisCurves ([page 146](#)) and GetAllCurves ([page 143](#))

curve_label New label for curve. To restore a label to its default value set this to %DefaultLabel%

Curve labels can also be edited using the SetGraphAnnoProperty (below) command to edit the curve's Label property.

SetGraphAnnoProperty

SetGraphAnnoProperty *object-id property-name property-value*

Sets a property value for a graph object. Note that this command's name is a little misleading as it can edit the values of the properties of any graph object not just *annotation* objects. For more information on graph objects and properties see “[Graph Objects](#)” on [page 448](#).

object-id Id of object which owns the property to be edited.

property_name Name of property to be edited

Property-value New value of property.

SetGroup

SetGroup *group_name*

Changes the current group.

group_name Name of new group. An array of current group names is returned by the Groups function ([page 210](#)).

See “[Groups](#)” on [page 210](#).

SetHighlight

SetHighlight /prop *propname* [*propvalue*]

SetHighlight /wire *wirehandle*

SetHighlight /all 1|0

SetHighlight /clearAllOpen

SetHighlight 1|0

Highlights or unhighlights schematic objects.

/prop propname propvalue

Property name. If specified without *propvalue* all instances possessing *propname* will be highlighted. Otherwise only instances possessing *propname* with *propvalue* will be highlighted.

/wire wirehandle

Handle of wire to be highlighted.

/all 1|0

If '1' specified, highlights all objects on selected schematic. Otherwise, unhighlights all objects on selected schematic.

/clearAllOpen

Clears all highlighting on all open schematics.

1|0 (last form)

If '1' all selected objects highlighted, otherwise all selected objects unhighlighted.

SetOrigin

SetOrigin *x y*

Sets the origin of the current symbol.

x, y

The co-ordinates of the origin in units of 100 per grid square. The origin is placed relative to a location defined by the top left of a rectangle that encloses all the pins of the symbol.

The symbol's origin is a reference point used to define the location of all the elements of the symbol. In the majority of applications the position of the origin is immaterial as long as it does not change once an instance of the symbol has been placed on a schematic. If a new symbol is created from scratch to replace an old one, its origin would have to be maintained and this command would be needed for this. In practice, however, the user would usually modify an existing symbol in which case the origin would be maintained automatically.

See Also

Function [“GetSymbolOrigin” on page 199](#)
[“SetSymbolOriginVisibility” on page 423](#)

SetReadOnly

SetReadOnly *vector-name*

Sets a vector to be read-only. Once so assigned a vector can not be written to. Note that this is a one-way operation. It is not possible to remove the read-only status of a vector.

This command is intended for use when the program starts (possibly called from the startup script) to assign values as constants which can never be changed or deleted.

SetRef

SetRef *vector_name* *reference_expression*

Attaches *reference_expression* to *vector_name*. Previous reference is detached and deleted if no longer used. See “Expressions” on page 30 for details on references.

See Also

“Expressions” on page 30.

SetSnapGrid

SetSnapGrid *snapgrid*

snapgrid Snap grid in sheet units. May be 120 (default), 60, 40, 30 or 24

Warning: only change the snap grid if there is no alternative. We strongly recommend against changing the snap grid simply to satisfy personal preferences as doing so may introduce compatibility problems, especially if applied to symbols.

Sets the snap grid for the currently selected schematic or symbol editor window. The snap grid is the grid on which wires and symbol pins lie. The default value is 120 but may be changed to 60, 40, 30 or 24. Note that this command will not allow the snap grid to be changed to something that would place existing wires or symbols off grid.

SetSymbolOriginVisibility

SetSymbolOriginVisibility show|hide|toggle

Controls the visibility of the origin marker in the symbol editor.

SetToolBarVisibility

SetToolBarVisibility *toolbar_name* *visibility*

Sets the visibility of a toolbar.

toolbarname Either a pre-defined toolbar (see “DefineToolBar” on page 362) or a new one created with `CreateToolBar` (page 355).

visibility Specifies when toolbar is visible. See “CreateToolBar” on page 355 for details.

See Also

“CreateToolButton” on page 356

“DefButton” on page 361

“GetToolButtons” on page 204

SetUnits

SetUnits *vector_name physical_type*

Changes physical type of *vector_name* to *physical_type*. Physical type may be any of the following:

'unknown'	'?'
'Voltage'	'V'
'Current'	'A'
'Time'	'Secs'
'Frequency'	'Hertz'
'Resistance'	'Ohm'
'Conductance'	'Sie'
'Capacitance'	'F'
'Inductance'	'H'
'Energy'	'J'
'Power'	'W'
'Charge'	'C'
'Flux'	'Vs'
'Volt^2'	'V^2'
'Volt^2/Hz'	'V^2/Hz'
'Volt/rtHz'	'V/rtHz'
'Amp^2'	'A^2'
'Amp^2/Hz'	'A^2/Hz'
'Amp/rtHz'	'A/rtHz'

" (means dimensionless - see notes)

The physical type of a vector is the name of the physical quantity it represents e.g. Voltage, Current, Time etc. This is used by graph plotting routines to set appropriate units for axes. To set a vector as dimensionless, use the following syntax:

```
SetUnits vector {''}
```

SetWireColour

SetWireColour 0|1

Schematic wires can be displayed in an alternative colour specified by the colour name 'Alt Wire Colour'. To edit this colour select menu File|Options|Colour.... Executing the command:

```
SetWireColour 1
```

will set all selected wires to use the alternative colour. The command

```
SetWireColour 0
```

will restore selected wires to the normal colour.

Shell

```
Shell [/wait] [/displayStdout] [/command command_string]
application_name
```

Launches an application.

<code>/wait</code>	If specified, application is launched synchronously. This means that SIMetrix will not continue until the application has closed.
<code>/displayStdout</code>	Displays in the message window any standard output from the program
<code>/command <i>command_string</i></code>	Ignored under Linux. Calls system command processor to execute <i>command_string</i> . This is necessary to run internal commands such as Copy and Move. The command processor is usually CMD.EXE
<i>application_name</i>	File system path to executable file. This would usually be a binary executable but may be any file that is defined as executable by the operating system. If a full path is not specified, a search will be made for the file using the rules described in “Shell” on page 290.

Notes

Note that under Windows, console mode applications will be launched without the console. This is for cross-platform compatibility as Linux and other operating systems do not support the concept of console mode.

To run a console mode application in a manner such that the console is displayed, use the ShellOld command (see below).

ShellOld

```
Shell [/wait] [/icon|hide] [/command command_string]
application_name
```

Launches an application. This is implemented only on Windows and behaves identically to the Shell command implemented on version 4.5 and earlier.

<code>/wait</code>	If specified, application is launched synchronously. This means that SIMetrix will
--------------------	--

	not continue until the application has closed.
<code>/command <i>command_string</i></code>	Calls system command processor to execute <i>command_string</i> . This is necessary to run internal commands such as Copy and Move. The command processor is usually CMD.EXE
<code>/icon hide</code>	If /icon, the program is started in a minimised state. If /hide, the program main window is initially hidden.
<code><i>application_name</i></code>	File system path to executable file. This would usually be a binary executable but may be any file that is defined as executable by the operating system. If a full path is not specified, a search will be made for the file using the rules described in “Shell” on page 290.

Show

Show [/file *filename*] [/append *filename*] [/noindex] [/noHeader]
[/plain] [/force] [/names *names*] [/clipboard] [/width *width*] [/lock]
expression [*expression* ...]

Displays the value of an expression.

<code>/file <i>filename</i></code>	If specified, outputs result to filename. The values are output in a format compatible with OpenGroup /text (page 394).
<code>/append <i>filename</i></code>	As /file except that file is appended if it already exists.
<code>/noindex</code>	If the vector has no reference, the index value for each element is output if this switch is <i>not</i> specified.
<code>/noHeader</code>	If specified, the header providing vector names etc. will be inhibited.
<code>/plain</code>	If specified, no index (as /noindex), and no header (as /noHeader) will be output. In addition, string values will be output less the quotation marks.
<code>/force</code>	File specified by /file will be unconditionally overwritten if it exists.
<code>/names <i>names</i></code>	Semicolon delimited list of column labels. If specified, each vector column will be labelled by the corresponding name given in <i>names</i> . Otherwise, vector name is used as label.
<code>/clipboard</code>	If specified, the result is copied to the windows clipboard.
<code>/width</code>	Page width in columns.
<code>/lock</code>	If specified with /file, a lock file will be created while the write operation is being performed. The file will have the extension .lck. This can be used to synchronise data transfers with other applications. Under Windows the file will be locked for write operations. On Linux the file will have a cooperative lock

applied.

expression Expression to be displayed. If *expression* is an array, all values will be displayed.

Notes

To enter multiple expressions, separate each with a comma.

The display of arrays with a very large number of elements (>500) can take a long time. For large arrays it is recommended that the `/file` or `/clipboard` switch is used to output the results to a file or the windows clipboard respectively. The data can then be examined with a text editor or spreadsheet program.

The results will be tabulated if all vectors are compatible that is have the same x-values. If the any vectors listed are not compatible, each vector's data will be listed separately.

The precision of numeric values can be controlled using the "Precision" option setting. Use the command "Set precision = *value*". This sets the precision in terms of the column width.

ShowCurve

ShowCurve

Shows specified curve having been hidden using HideCurve

curve_id Id of curve to show. Curve id is returned by the functions GetSelectedCurves ([page 187](#)), GetAxisCurves ([page 146](#)) and GetAllCurves ([page 143](#))

See Also

HideCurve [page 381](#)

ShowSimulatorWindow

ShowSimulatorWindow

Displays simulator status window if it is currently hidden.

SizeGraph

SizeGraph [`/xfull`] [`/yfull`] [`/axisid axis_id`] *xoffset* *yoffset* *xscale* *yscale*

General purpose command to zoom or scroll a graph.

`/xfull` If specified, the x-axis is zoomed to fit whole graph. *xscale* and *xoffset* will be ignored

`/yfull` If specified, the y-axis is zoomed to fit whole graph. *yscale* and *yoffset* will be ignored

Script Reference Manual

<i>/axisid axis_id</i>	Specify which y-axis to resize. If omitted, all y-axes on selected graph will be affected.
<i>xoffset</i>	Extent of X-shift as proportion of full width of graph. E.g. 0.25 will shift by a quarter. 0 has no effect.
<i>yoffset</i>	As <i>xoffset</i> but for y-axis
<i>xscale</i>	View width required as proportion to current width. E.g. 0.8 will zoom in by 20%. 1 has no effect. 0 is illegal.
<i>yscale</i>	As <i>xscale</i> but for y-axis.

Stats

Stats

Displays statistics relating to most recent simulation.

Note that time values other than total analysis time will not be supplied unless the simulator option “TIMESTATS” or “ACCT” is specified.

TemplateEditProperty

TemplateEditProperty *ref propname propvalue*

This command may only be executed in a template script. Please see “[Schematic Template Scripts](#)” on page 464 for more information. In other situations use the Prop (page 402) command.

Edits the property of a schematic instance.

<i>ref</i>	Component reference of instance to be edited. This would usually be the REF value passed to the template script
<i>propname</i>	Name of property to be changed
<i>propvalue</i>	New value for property

TemplateSetValue

TemplateSetValue *ref template_value*

This command may only be executed in a template script. Please see “[Schematic Template Scripts](#)” on page 464 for more information.

Sets the value that will be used for the specified device’s template during the current netlist operation. Note that this command does *not* change the value of the TEMPLATE property stored on the instance itself.

<i>ref</i>	Component reference of instance. This would usually be the REF value passed to the template script.
<i>template_value</i>	Template value

TextWin

TextWin hide|show|toggle

Hide/Show the schematic's text window (also known as the "F11 Window") for entering simulator controls.

TextWin toggle will hide the text window if it is currently visible and vice-versa.

Title

Title schem|graph *new_title*

Changes a window's title.

schem	Apply to selected schematic window
graph	Apply to selected graph window
<i>new_title</i>	New window title

Notes

The title is displayed in the window's caption bar and is also placed at the bottom of printed graphs and schematics.

Trace

Trace *signal_name trace_id*

The trace command is used to set up a simulation trace while a simulation is running. To set up a trace before a simulation is started, use the .TRACE or .GRAPH simulator controls.

<i>signal_name</i>	Net name or pin name for voltage or current to be traced.
<i>trace_id</i>	Integer value used to group traces together on the same graph. All traces with the same <i>trace_id</i> will go to the same graph.

Note that traces set up with this command only remain in effect until the end of the simulation. A Trace command executed before a simulation starts will have no effect.

Undo

Undo

Undo the last schematic editing operation.

See Also

Redo [page 408](#)

UndoGraphZoom

UndoGraphZoom

Restores previous graph view area. Successive execution of this command will retrace the entire history of graph magnification and scroll positions.

UnHighlightCurves

UnHighlightCurves

Unhighlights all curves.

UnLet

Unlet *vector_name*

Destroy vector.

vector_name Name of vector to be destroyed. Unless the vector is in the *user* group, the vector's full qualified name must be used.

See Also

[“Expressions” on page 30](#)

[“Let” on page 385](#)

Unprotect

Unprotect

Unprotects and selects protected schematic components.

See Also

[“Protect” on page 405](#)

Unselect

Unselect

Unselects all components and wires on selected schematic.

UnSet

UnSet [/temp] *option_name*

Deletes specified option. See [“Set” on page 420](#)

/temp Deletes only temporarily. Will revert to original value once control returns to the command line.

Note that some Option values are *internal*. This means that they always have a value. If such an option is UnSet, it will be restored to its default value and not deleted.

UpdateAllSymbols

UpdateAllSymbols

Checks all symbols in all open schematics and updates them if they are defined with the “All references to symbol automatically updated” flag is set in the library symbol definition.

It isn’t usually necessary to call this command. It is automatically called in any situation where changes might result from it.

UpdateProperties

UpdateProperties [/all] [*property-name* [*property-value*]]

Restores the properties on a schematic instance to values defined by its symbol.

/all If specified, all instance properties will be restored to their symbol defined values. Also properties will be deleted or added so that the instance matches the symbol in every way. In short, the instance is left in the state it would be in if it had just been placed using the Inst command.

property-name property-value

Defines which instance(s) to operate on. If absent, operation will be performed on all selected instances. If *property-name* present but not *property-value* operation will be performed on all instances possessing *property-name*. Otherwise the operation will be performed on all instances possessing *property-name* with a value of *property-value*.

UpdateSymbol

UpdateSymbol *symbol_name*

Updates symbols on currently selected schematic from global symbol library.

symbol_name Name of symbol to be updated.

Schematics store local copies of any symbols that it uses. If the copy of that symbol in the global library is modified, the schematics own copy is unaffected. This command causes the specified symbol to be updated from global library. See [“How Symbols are Stored” on page 445](#).

ViewFile

ViewFile *filename*

Opens a read only file viewer with specified file name. The file viewer is internal while the file editor called by EditFile is an external program.

Wait

Wait

Suspends command execution until any mouse key is clicked. Wait does not suspend commands executed directly on assignment to keystrokes or menu items. This allows the cancel command, when assigned to a key or menu, to terminate a wait command.

Where

This command has been deleted. Instead an internal script of the same name exists, which performs the same task as the original Where command.

The Where script displays convergence information about the most recent run.

Wire

Wire [/start] [/loc x1 y1 x2 y2] [/mode]

Enter schematic wiring mode.

/start	If specified, a new wire is started.
/loc x1 x2 y1 y2	If specified, command in non-interactive and wire is placed at location specified by x1 y1 x2 y2. Co-ordinates are relative and would usually be derived from a call to WirePoints (page 319).
/mode	If specified, the schematic editor is placed in a temporary wiring mode. The next left click will start a wire and wiring may proceed in the usual manner. After pressing the right mouse button, wiring mode will be cancelled.

WireMode

WireMode on|off

Switches schematic wiring mode on or off.

WriteImportedModels

WriteImportedModels *netlist filename*

Writes all library models required by *netlist* to *filename*.

Zoom

Zoom rect|full|out|in

Changes magnification of currently selected schematic.

rect	User selects area to be viewed with mouse.
full	Fits schematic to the current window

out Magnification is reduced one step
in Magnification is increased one step

Chapter 7 Applications

User Interface

A full description of the user interface is outside the scope of this manual. Instead, in this section, we provide a few pointers on how to go about finding how a particular feature works so that it can be altered or adapted.

User Defined Key and Menu Definitions

Virtually the entire user interface is accessed through menus, keyboard keys or toolbar buttons all of which may be redefined, deleted or replaced. The only parts of the UI which are not accessible are the mouse keys. These have fixed definitions and may not be modified by the user.

In principle it is possible to define completely new menus or/and toolbars which bear no similarity with the built-in definitions. A more normal use of menu, button and key redefinition would probably be to add a special function or perhaps to delete some unused items.

Menus are defined using the command [DefMenu \(page 368\)](#) and keys can be defined with the command [DefKey \(page 365\)](#). To define toolbars and buttons, see [Creating and Modifying Toolbars \(page 466\)](#). Commands to define new user interface elements such as menus are usually placed in the [Startup Script \(page 47\)](#).

Key definitions may be *context sensitive*. That is, the definition is dependent on which type of window is currently active.

Rearranging or Renaming the Standard Menus

The standard menu definitions are loaded from the built in script 'menu' when the program first starts. The source for all built in (or internal) scripts can be found on the install CD the latest version of which may be downloaded from our web site (<http://www.simetrix.co.uk>). To modify any of the standard menus, you need to modify the 'menu' script. For details on how to modify internal scripts, see "[Modifying Internal Scripts](#)" on [page 435](#)

When editing menu.sxsr, please note the following:

- Each menu definition must occupy a single line
- Menus are created in the order they appear in the script. To change the order, simply rearrange the lines.
- You can disable a menu definition by putting a '*' as the first character of the line. This makes it easy to later undelete it.

Menu Shortcuts

These are keys which activate defined menus. The key name is displayed to the right of the menu text. All menu definitions may have shortcuts specified using the "/shortcut" switch for the DefMenu command. A potential problem arises if the same key is used

for a shortcut and a key definition using DefKey. If this happens, the DefKey definition takes precedence.

Editing Schematic Component Values

When you press F7 or select the schematic popup menu Edit Value/Model the internal script 'value' is called. 'value' is a complicated script that identifies the type of component that is selected and performs an action appropriate for it. However the first thing this script does is find out if the component (or components) selected have a *valuescript* property. If it does then that script is called. This feature is used by all types of component developed since release 3 but some older devices are handled differently.

If you wish to modify the behaviour for a particular component type when F7 is pressed, first check to see if it has a *valuescript* property. If it has you can edit the script that it calls or change the property's value to call a different one. If it hasn't you can add such a property and provide a script for it.

There are two other properties associated with component values. These are *incscript* and *decscript*. These increment and decrement a components value when the shift-up and shift-down keys are pressed. Currently only the resistors, capacitor, inductor and potentiometer symbols use this property but you can add your own to any other symbol.

Modifying Internal Scripts

The SIMetrix user interface is implemented with about 550 internal (or built-in) scripts. These are built in to the executable file but can be bypassed so that their function can be changed. The code for all of these scripts can be found on the installation CD in directory script/builtin. The procedure for replacing an internal script is very straightforward. Simply place a script with the same name but with the extension .sxscr in the built-in script directory. The location of this directory is set in the file locations sheet of the options dialog box (menu File|Options|General...). On Windows this is usually <SIMetrix root>\support\biscript and on Linux it is <SIMetrix root>/share/biscript. SIMetrix always searches this directory first when executing an internal script.

Custom Curve Analysis

The menus Probe|More Probe Functions... and the graph menu Measure|More Functions... each open a tree list dialog box that displays the function available. In this section we describe how this system works and how it can be extended.

We have only skimmed over the basics. For more information, please refer to the scripts themselves.

Adding New Functions

The operations listed for the menus described above are obtained from one of two built-in text files. These files are:

analysis_tree.sxscr	For curve analysis functions
probe_tree.sxscr	For probe functions

Like built in scripts, these are embedded in the binary executable but can also be overridden by placing files of the same name in the biscript directory.

Both files use the same format. Each entry in the tree list is defined by a single line in the file. Each line contains a number of fields separated by semi-colons. The first field is that command that is called to perform the action while the remaining fields describe the hierarchy for the entry in the tree list control. The command is usually a script often with a number of arguments. To add a new function, simply add a new line to the relevant file. The order is not important.

'measure', 'measure_span' Scripts

These are the “driver” scripts that perform the curve analysis and curve analysis over cursor span analysis respectively. These don't perform the actual calculations but carry out a number of housekeeping tasks. The calculations themselves are performed by a script whose name is passed as an argument. To add a new function you need to create one of these scripts. For simple functions the script is not complicated. In the example below we show how the “Mean” function is implemented and you will see that it is very simple.

An Example: The 'Mean' Function

The entry for the full version of this in `analysis_tree.txt` is:

```
measure_mean;Measure;Transient;Mean;Full
```

This means that the script 'measure_mean' will be called when this function is selected. 'measure_mean' is quite simple, it is just a single line

```
measure /ne 'calculate_mean' 'Mean'
```

`/ne` is not that important, it just tells the script system not to enter the command in the history list.

'calculate_mean' specifies the script to call to perform the calculation.

'Mean' specifies the y-axis label

The 'calculate_mean' script is as follows:

```
Arguments data xLower xUpper @result @error

if xUpper>xLower then
  Let result = Mean1(data, xLower, xUpper)
else
  Let result = Mean1(data)
endif
```

The argument `data` is the data that is to be processed. In this case we simply need to find its Mean. `xUpper` and `xLower` specify the range over which the mean should be calculated. These would be specified if the “cursor span” version of the mean function was selected by the user. The result of the calculation is assigned to the argument `result` which has been “passed by reference”. The `error` argument is not used here

but it can be used to signal an error condition which will abort the operation. This is done by setting it to 1.

Automating Simulations

Overview

The script language allows you to automate simulations, that is automatically run a number of simulation runs with different component values, test conditions or analysis modes. This section describes the various commands needed to do this.

Running the Simulator

Simulations are started using the Run command ([page 410](#)). The Run command runs a netlist not a schematic, so you must first create the netlist using the NetList command ([page 390](#)). Some notes about the Run command:

1. The /an switch is very useful and allows you to run different analyses on the same circuit without having to modify it. /an specifies the analysis mode instead and overrides any analysis controls (e.g. .TRAN, .DC etc.) in the circuit itself.
2. If the run fails (e.g. due to non-convergence), the script will abort without performing any remaining runs. This behaviour can, however, be inhibited with the /noerr switch which must be placed immediately after the Run word:

```
Run /noerr /file design.net
```

/noerr is a general switch that can be applied to any command. See “[Command Switches](#)” on [page 34](#) for details. If you want to test whether or a run was successful, use the GetLastError function ([page 172](#)).

Changing Component Values or Test Conditions

It is likely that in an automated run you will want to change component values or stimulus sources between runs. There are a number of ways of doing this, each with its own advantages and disadvantages.

Edit Schematic

With this method, the changes are made to the schematic which is then re-netlisted. To do this you need to become familiar with the commands Prop ([page 402](#)), Select ([page 419](#)) and Unselect ([page 430](#)). The procedure is first unselect everything, then select the component you wish to change and then use the Prop command to change the value. The following will change the value of R5 to 12k:

```
Unselect
Select /prop Ref R5
Prop value 12k
```

The second line says “select the component with a Ref property of R5”. The third line says “change the value property of the selected component(s) to 12k”.

You use the same basic method to edit a stimulus. The following sets V1 to be a pulse source with 0V start, 5V end, zero delay 10nS rise and fall times, 1 μ S pulse width and 2.5 μ S period.

```
Unselect
Select /prop Ref V1
Prop value "Pulse 0 5 0 10n 10n 1u 2.5u"
```

Note the quotation marks.

You must ensure that you re-netlist the circuit before running the simulation.

Circuit Parameters

Rather than edit the schematic and re-netlist, an alternative is to specify the component values as parameters then vary the parameter using the Let command. To do this, you must first edit the value of the components to be varied so that they are represented as a parameter expression enclosed by curly braces '{' and '}'. Again we will use the example of a resistor R5 whose value we wish to set to 12K. Proceed as follows:

1. Select R5 then press *shift*-F7. Enter {R5} as the new value.
2. Now in the script you can set the value of R5 with Let e.g.

```
Let global:R5=12k
```

The global: prefix is necessary to make the parameter global. Note we have named the parameter 'R5'. This is an obvious choice of parameter name but you could use anything as long as it starts with a letter and consists of letters numbers and the underscore character. (You *can* use other characters but we don't recommend it).

You must use curly braces when defining parameters in this manner. Expressions enclosed in quotation marks will not evaluate if they access global parameters. You can however define another parameter using .PARAM which will be accessible in quoted expressions. E.g.

```
.PARAM local_R5={R5}
```

local_R5 as defined above will be accessible in any type of expression in the netlist.

Expressions in curly braces that consist entirely of global parameters or/and constants and which have no local (.PARAM defined) parameters, may also be used to define simulator control values as well as component values. E.g.

```
.TRAN {stop_time}
```

is permissible as long as stop_time is defined using the Let command in a script.

An alternative, and somewhat more sophisticated approach is to change the component value to parameter version (e.g. "{R5}") in the script itself. You could then call Netlist to create the netlist with parameterised values after which the components can be restored to their original values. That way the schematic is preserved with its original values. To do this correctly you would need to save the original values so that they can

be restored. This can be done using the PropValue function ([page 259](#)) which returns the value of a property. The example shown below uses this technique.

Multiple Netlists

Conceptually this is probably the simplest approach but not very flexible. Simply create multiple versions of the netlist manually with different file names then run them one at a time.

Include Files

A method of making complex changes to a netlist is to incorporate part of it in a separate file and include it in the main netlist using the .INCLUDE simulator control. A script can then generate the lines in the include file. This can be done using the command [Show \(page 426\)](#) with the switch /plain to write a string array to a file. The string array can be created using the function [MakeString \(page 231\)](#) and built using custom code.

Organising Data Output from Automated Runs

A feature is available to organise data from multiple automated runs in the same way as for multi-step runs i.e. in the form of multi-division vectors. This is explained in the section describing the command [Run \(page 410\)](#).

An Advanced Example - Reading Values from a File

In this section we supply an example of quite an advanced script which runs several simulations on a circuit. On each run a number of components are changed to values read in from a file. This script is general purpose and can be used for any circuit without modification.

The script is quite complicated but is well commented throughout to explain in detail how it works. The basic sequence is as follows:

1. Get configuration file name from user
2. Read first line of file. This has the names of the components to be modified
3. Temporarily edit the modifiable components' values to reference a parameter
4. Create netlist
5. Restore original values
6. Read the rest of the file and write the values for each run to an array
7. Run the simulations
8. Clean up before exit

Here is the script. It is also supplied on the install CD under the script directory.

```
** Script to run multiple simulations using component values  
** read from a file  
  
** First ask the user for a file  
Let filename = GetSIMatrixFile('Text', ['open', 'all'])
```

Script Reference Manual

```
if Length(filename)=0 then
    ** User cancelled box
    exit script
endif

** Read the file
Let lines = ReadFile(filename)
Let numLines = Length(lines)

** Test it has enough lines
if numLines<2 then
    Echo "Definition file must have at least two lines"
    exit script
endif

** We now parse the file and read in the component values
** to the array "compValues". We do the whole file at the
** beginning so that the user will know straight away if it
** has any errors.

** The first line is the list of components that will be
changed

Let components=Parse(lines[0])
Let numComponents = Length(components)

if numComponents=0 then
    Echo "No component names specified"
    Echo "or first line of config file empty"
    exit script
endif

** Before we read the rest of the file, we will attempt to
** replace the values of all listed components with parameters
** and netlist the circuit. If any of the components don't
** exist then we will find out here.

** array to store original values so that we can restore
** them later
Let origValues = MakeString(numComponents)

Unselect
Let error = 0
** Scan through list of components
for idx = 0 to numComponents-1

    ** Select it
    Select /prop ref {components[idx]}

    if SelectCount()=0 then
        ** Select count is zero so select failed.
        ** This means the circuit doesn't have this component
        ** Output a message and set error flag.
        Echo "Cannot find component " {components[idx]}
        Let error = 1
    else
```

```

if HasProperty('value') then
    ** Save original value to be restored later
    Let origValues[idx] = PropValue('value')

    ** Set value as a parameter of name which is the
same
as the ref
    Let newVal = "'{' & PropValue('ref') & '}'"
    Prop value {newVal}
else
    ** The component does not have a value
property to alter.
    Echo "Component " {components[idx]}
    Echo "does not have a value"
    Let error = 1
endif
endif
Unselect

next idx

** We have changed all the components so now we can netlist
the circuit
if NOT error then
    Netlist design.net
endif

** Once we have the netlist we can restore the original values
Unselect
for idx = 0 to numComponents-1
    Select /prop ref {components[idx]}

    if SelectCount()<>0 then
        if HasProperty('value') then
            Prop value {origValues[idx]}
        endif
    endif

    Unselect
next idx

** If we had an error we must now abort
if error then
    exit script
endif

** Now read the rest of the file.
Create an array large enough to hold all the values.
The values are actually stored as strings.
That way we can vary
model names as well as values.
Let compValues = MakeString(numComponents*(numLines-1))
Let error = 0
Let resIdx=0
for lineIdx=1 to numLines-1

    ** Parse the line into individual values
    Let vals = Parse(lines[lineidx])

```

```
if Length(vals)<>numComponents then
    ** A line found with the wrong number of values.
    ** This is assumed
    ** to be a mistake unless the line is completely empty
    if Length(vals)<>0 then
        Echo "Wrong number of values at line " {lineIdx}
        Let error = 1
    endif
else
    ** line is OK so write the values to compValues
    for idx=0 to numComponents-1
        Let compValues[resIdx*numComponents+idx]=vals[idx]
    next idx
    ** Because some lines may be empty we have to use
    ** a different index counter for the compValues entries
    Let resIdx = resIdx+1
endif
next idx

if error then
    exit script
endif

** resIdx finishes with the number of non-blank data lines
Let numRuns = resIdx

** Now, at last, we can run the circuit
for idx=0 to numRuns-1

    for compIdx=0 to numComponents-1
        Let paramName = 'global:' & components[compIdx]
        Let {paramName}= compValues[idx*numComponents+compIdx]
    next compIdx

    Run /file design.net
next idx

** This isn't essential, but it is always best to delete
** global variables when we are finished with them
for compIdx=0 to numComponents-1
    Let paramName = 'global:' & components[compIdx]
    UnLet {paramName}
next compIdx
```

Schematic Symbol Script Definition

It is possible to define a schematic symbol using a script. This method is used in some of the internal scripts to create dynamic symbols. For example the transformer devices allow the user to define the number of both primary and secondary windings. The symbols for these are not stored in the symbol library but generated programmatically using the commands described in this section.

Symbol scripts can also be useful to edit symbols using a text editor. Some operations can be more rapidly performed by editing a text definition than by using a graphical editor. To support this method, SIMatrix includes the MakeSymbolScript command ([page 387](#)) that writes a script definition of a symbol in ASCII form.

The following sections describe how to define a symbol using a script.

Defining New Symbol

To define a new symbol (as opposed to modifying an existing one) proceed as follows:

1. Enter the text definition as described in “[Symbol Definition Format](#)” on page 443 into a text file (using NOTEPAD for example)
2. Load the new definition by simply typing the name of the file at the command line
3. Test that your new symbol is as you expect. Use the menu Place|From Symbol Library to place your symbol on a schematic

Note that as the schematic stores its own copy of each symbol, if you modify the symbol after first defining it, the changes will not be reflected in any existing schematics unless the “track” flag is set. This is done by providing the switch “/flags 1” on the CreateSym command line. This performs the same function as the “All references to symbol automatically updated” check box in the symbol editor save symbol dialog box.

To update the symbol on a schematic from the global library use the popup menu Update Symbols.

Symbol Definition Format

The following commands are used to define schematic symbols

AddArc [page 338](#)

AddCirc [page 339](#)

AddPin [page 343](#)

AddProp [page 344](#)

AddSeg [page 346](#)

CreateSym [354](#)

DelSym [page 374](#)

EndSym [page 379](#)

To describe the symbol definition format consider the definition for the npn transistor supplied in the standard symbol library. In text form this is:

```
* NPN BJT
CreateSym npn "NPN bipolar" analog
AddSeg 0 0 0 200
AddSeg 0 100 100 0
AddSeg 0 100 100 200
AddSeg 100 200 80 160
AddSeg 100 200 60 180
AddSeg 0 100 -100 100
AddPin C 1 100 0
AddPin B 2 -100 100
AddPin E 3 100 200
AddProp ref Q? 26
AddProp value NPN_MODEL 26
AddProp model Q 64
EndSym
```

Script Reference Manual

Let's go through it line by line.

The first line:

```
* NPN BJT
```

is a comment. Any text may placed after a '*' as the first character will be ignored.

The next line:

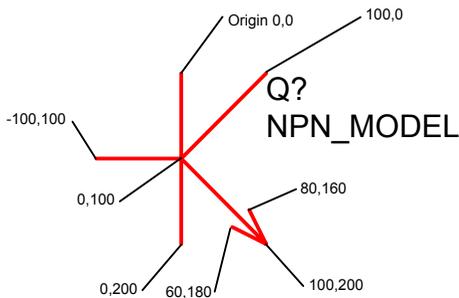
```
CreateSym npn "NPN bipolar" analog
```

begins the symbol definition. The first argument - npn - is the symbol name. This must be unique and cannot contain spaces. It is used to place the symbol on a schematic. The second argument is the description and is optional. This is what will appear in the *choose symbol* dialog box opened by the schematic popup Place|From Symbol Library... menu item. If no description is given the symbol's name will appear in this dialog box. The final parameter is the catalog name. This is used to categorise symbols. Although the parameter is optional, it is strongly recommend that it is included.

The following six lines:

```
AddSeg 0 0 0 200
AddSeg 0 100 100 0
AddSeg 0 100 100 200
AddSeg 100 200 80 160
AddSeg 100 200 60 180
AddSeg 0 100 -100 100
```

describe the symbol's six straight line segments. The four numbers on each line are the x and y co-ordinates of the start and end of each segment. 100 units represents one grid square (at X1 mag) on the schematic. The diagram shows the co-ordinates of each segment end.



The next three lines:

```
AddPin C 1 100 0
AddPin B 2 -100 100
AddPin E 3 100 200
```

describe the location and attributes of the symbol's three pins. The first parameter on each AddPin command is the pin's name. This must be the same as the pin name used

by the simulator for that type of device. If the name is different it will not be possible to cross-probe currents for that type of device. See the section “Summary of Simulator Devices” in the User’s manual for details of pin names for all devices supported by the simulator. If the device is a subcircuit then any pinname may be used. The second parameter is the pin’s number. This affects the order in which the pin’s connected nets appear in the netlist. This must comply with the netlist format. Again refer to “Summary of Simulator Devices” for full details of each device. The last two parameters specify the co-ordinates of the pins on the schematic. They *must* be a multiple of 100. If they are not it will not be possible to connect to them as wire ends *always* snap to a grid point. See AddPin (page 343) for more details.

The next three lines:

```
AddProp ref Q? 26
AddProp value NPN_MODEL 26
AddProp model Q 64
```

are the symbol’s properties. A symbol’s component reference, value (or model name) and the type of device are all specified by properties. The first line above attaches a “ref” property (aka component reference) and gives it an initial value of Q?. The final parameter ‘26’ specifies how it should be displayed on the schematic. The model property in the third line specifies the type of device (e.g. resistor, capacitor, BJT etc.) and is always a single letter. It is not compulsory. If it is omitted the first letter of the ref property is used instead. See “Summary of Simulator Devices” for full list of devices supported by the simulator and their required model properties. Full details on properties are given in the User’s manual. See page 344 for documentation for the AddProp command.

The final line:

```
EndSym
```

terminates the model definition. The symbol will not be recognised until this is executed.

How Symbols are Stored

Symbol definitions are first stored in a .sxslib file which resides in the SymbolLibs directory. These files are managed by the symbol library manager. When a symbol is placed on a schematic, a copy of that symbol definition is stored in the schematic and from then on the schematic will use that copy of it. This means that if you change a symbol definition for a schematic that is saved, when you open that schematic, it *may* still be using the old definition as it is saved with the schematic. Whether or not the symbol is updated automatically depends on how it was saved. If “/flags 1” was included with the CreateSym command, then it will be automatically updated.

If you wish to force the schematic to use the new symbol, select the symbol or symbols then select the popup menu Update Symbols. Note that all instances of the symbol will be updated. It is not possible to have two versions of a symbol on the same schematic.

Data Import and Export

This section is also in the User’s manual. It is reproduced here for convenience.

SIMetrix provides the capability to export simulation data to a file in text form and also to import data from a file in text form. This makes it possible to process simulation data using another application such as a spreadsheet or custom program.

Importing Data

To import data use the OpenGroup command ([page 394](#)) with the /text switch. E.g. at the command line type:

```
OpenGroup /text data.txt
```

This will read in the file data.txt and create a new group called text*n*. See “Data Files Text Format” below for details of format.

Note that if you create the file using another program such as a spreadsheet, the above command may fail if the file is still open in the other application. Closing the file in the other application will resolve this.

Exporting Data

To export data, use the Show command ([page 426](#)) with the /file switch. E.g

```
Show /file data.txt vout r1_p q1#c
```

will output to data.txt the vectors vout, r1_p, and q1#c. The values will be output in a form compatible OpenGroup /text.

Vector Names

In the above example the vector names are vout, r1_p and q1#c. If you simulate a schematic, the names used for voltage signals are the same as the node names in the netlist which in turn are assigned by the schematic's netlist generator. To find out what these names are, place the mouse cursor on the node of interest on the schematic then press cntrl-S. The node name - and therefore the vector name - will be displayed in the command shell. A similar procedure can be used for currents. Place the mouse cursor on the device pin of interest and press cntrl-P.

Launching Other Applications

Data import and export makes it possible to process simulation data using other applications. SIMetrix has a facility to launch other programs using the Shell command. You could therefore write a script to export data, process it with your own program then read the processed data back in for plotting. To do this you must specify the /wait switch for the Shell command to force SIMetrix to wait until the external application has finished. E.g.

```
Shell /wait procddata.exe
```

will launch the program procddata.exe and will not return until procddata.exe has closed.

Data Files Text Format

There are two alternative formats.

The first is simply a series of values separated by whitespace. This will be read in as a single vector with a reference equal to its index.

The second format is as follows:

A text data file may contain any number of *blocks*. Each *block* has a *header* followed by a list of *datapoints*. The *header* and each *datapoint* must be on one line.

The *header* is of the form:

```
reference_name ydata1_name [ ydata2_name ... ]
```

Each *datapoint* must be of the form:

```
reference_value ydata1_value [ ydata2_value ... ]
```

The number of entries in each *datapoint* must correspond to the number of entries in the *header*.

The *reference* is the x data (e.g. time or frequency).

Example

Time	Voltage1	Voltage2
0	14.5396	14.6916
1e-09	14.5397	14.6917
2e-09	14.5398	14.6917
4e-09	14.54	14.6917
8e-09	14.5408	14.6911
1.6e-08	14.5439	14.688
3.2e-08	14.5555	14.6766
6.4e-08	14.5909	14.641
1e-07	14.6404	14.5905
1.064e-07	14.6483	14.5821

If the above was read in as a text file (using `OpenGroup/text`), a new group called `textn` where `n` is a number would be generated. The group would contain three vectors called `time`, `"Voltage1"` and `"Voltage2"`. The vectors `"Voltage1"` and `"Voltage2"` would have a *reference* of `"Time"`. `"Time"` itself would not have a *reference*.

To read in complex values, enclose the real and imaginary parts in parentheses and separate with a comma. E.g:

```
Frequency : VOUT
1000 (-5.94260997 ,0.002837811 )
1004.61579 (-5.94260997 ,0.00285091 )
1009.252886 (-5.94260996 ,0.002864069 )
1013.911386 (-5.94260995 ,0.002877289 )
1018.591388 (-5.94260994 ,0.00289057 )
1023.292992 (-5.94260993 ,0.002903912 )
1028.016298 (-5.94260992 ,0.002917316 )
1032.761406 (-5.94260991 ,0.002930782 )
1037.528416 (-5.9426099 ,0.00294431 )
1042.317429 (-5.94260989 ,0.0029579 )
1047.128548 (-5.94260988 ,0.002971553 )
```

Graph Objects

Overview

Graph objects are the items displayed in a graph window. These include curves, axes, cursors and the various objects used for annotation. All graph objects possess a number of named properties all of which may be read and some may also be written. Each graph object also has a unique id which is used to identify it.

A knowledge of the inner workings of graph objects will be useful if you wish to customise some of the annotation features provided by the waveform viewer. However, the interface is at a low level with much work carried out by internal scripts. Consequently there is quite a steep learning curve to climb in order to make good use of the features available.

Object Types

The following table lists all the available object types

Object name	Description
Axis	Axes and grids
Crosshair	Crosshair part of cursor
CrosshairDimension	Object used to dimension cursors. Forms part of cursor. Cannot be displayed on its own
Curve	Curve
CurveMarker	Marker used to annotate curves
FreeText	Free Text annotation object. Displays unboxed text on graph
Graph	Graph sheet
LegendBox	Box enclosing LegendText objects
LegendText	Text objects used in legend boxes and linked to a displayed curve.
TextBox	Box enclosing FreeText object

Properties

Properties are the most important aspect of graph objects. Each type of graph object possesses a number of properties which determine characteristics of the object. Some properties are read only and are either never altered or can only be altered indirectly. Other properties can be changed directly using the command `SetGraphAnnoProperty` (see [page 421](#)). The labels for curves, axes and the various annotation objects are examples of properties that may be edited.

A full list of all object types and their properties is given in “[Objects and Their Properties](#)” starting on [page 450](#).

Graph Object Identifiers - the “ID”

Each instance of a graph object is uniquely identified by an integer value known as its “ID”. Valid IDs always have a value of 1 or greater. IDs are returned by a number of functions (see below) and also a number of the objects possess properties whose value is the ID of a related object.

Once the ID of an object has been obtained, its property names can be read and its property values may be read and/or modified.

The following functions return graph object IDs. Note that all functions return object IDs belonging to the currently selected graph only except for `GetGraphObjects` which can optionally return IDs for objects on a specified graph.

AddGraphCrossHair (page 74)	Add a new cursor to the currently selected graph and return its and its dimension's IDs
GetAllCurves (page 143)	Returns the IDs for all curves
GetAllYAxes (page 144)	Returns the IDs for all Y-axes
GetAxisCurves (page 146)	Returns IDs for all curves attached to specified y-axis
GetCurrentGraph (page 151)	Returns the ID for the currently selected graph sheet
GetCursorCurve (page 151)	Returns information about curve attached to the main cursor including its ID
GetCurveAxis (page 152)	Returns ID of y-axis associated with a curve.
GetDatumCurve (page 153)	Returns information about curve attached to the reference cursor including its ID
GetGraphObjects (page 164)	Returns all objects on a graph, or objects of a specified type
GetSelectedCurves (page 187)	Returns IDs of all selected curves
GetSelectedGraphAnno (page 187)	Returns ID of the selected annotation object
GetSelectedYAxis (page 188)	Returns the ID of the selected Y-axis
GetXAxis (page 209)	Returns the ID of the X-axis

Some of the functions in the above list are technically redundant. For example the value obtained by `GetCurveAxis()` can also be obtained by reading the value of the 'Y-axis' property of the curve. This can be done with the general purpose `GetGraphObjPropValue` ([page 165](#)) function.

Symbolic Values

Some properties used for labels may be given symbolic values. Symbolic values consist of a property name enclosed with the '%' character. When the label is actually displayed the property name is replaced with its value.

Symbolic values may also be indirect. Some properties return the id of some other associated object and the value of a property for that object may be referenced with a symbolic value. The '!' character is used to denote indirect symbolic values. For example, this method is used with curve markers. The default value for a curve marker's label is:

```
%curve:label%
```

“curve” is a property of a curve marker that returns the id of the curve that it points to. “label” is a property of a curve that returns the label assigned to it. So “curve:label” returns the label of the curve that the curve marker points to.

Other curve properties can be used for this label. For example, curve measurements (as displayed below the legend in the legend panel) can also be accessed via property named “measurements”. So the curve marker label:

```
%curve:label% %curve:measurements%
```

would display the curve's name followed by its measurements.

Finally the character sequence <n> can be used to denote a new line.

Objects and Their Properties

The following lists all the properties available for all objects. Note that all objects have a 'Type' property that resolves to the object's type name. Also all objects except Graph have a 'Graph' property that returns the ID of the object's parent graph sheet.

Axis

Axis objects represent both x and y graph axes

Name	Description	Read Only?
Type	Type of object - always 'Axis'	Yes
Graph	ID of parent graph	Yes
AxisType	'X', 'Y' or 'Dig'	Yes
Label	Label used to annotate axis. (Actual displayed text is <label> / <unit>). Default = %DefaultLabel%	No
Name	Axis name. ('Y1', 'Y2' etc.). Empty for X and Dig axes	Yes
Unit	Physical units of axis. (E.g. 'V', 'A' etc.). Default = %DefaultUnit%	No
Min	Minimum limit of axis	No
Max	Maximum limit of axis	No

Name	Description	Read Only?
AutoLimit	'On' or 'Off' determines whether axis limits are adjusted automatically according to attached curves	No
Grad	Grading of axis: "Log" or "Lin".	No
Delta	Value that determines the minor grid line spacing	No
VertOrder	Vertical order. Arbitrary string used to specify vertical display order	No
DefaultLabel	Label property is given default value of %DefaultLabel% which resolves to the value of this property.	Yes
DefaultUnit	Unit property is given default value of %DefaultUnit% which resolves to the value of this property.	Yes

Crosshair

Object used to display cursor. Each graph cursor consists of a Crosshair and two CrosshairDimensions.

Name	Description	Read Only?
COM1	Common reference value. Only meaningful with X-Y plots such as Nyquist plots. Shows the value of the common reference to X and Y. This is frequency in a Nyquist plot	Yes
Dimensions	Comma delimited string providing the dimension objects attached to this cursor	Yes
Graph	ID of parent graph	Yes
ID	ID of this object	Yes
Point	1 = Main cursor. 2 = Reference cursor	Yes
Type	Type of object - always 'Crosshair'	Yes
XDimension	The ID for the CrosshairDimension object that displays the X dimension and positions	Yes
YDimension	The ID for the CrosshairDimension object that displays the Y dimension and positions	Yes
Curve	ID of attached curve	No
Division	Division index of attached curve. See page for details on multi-division vectors	No
Frozen	'On' or 'Off'. If 'On' the user will not be able to move the cursor with the mouse	No
Hidden	Cursor is hidden	No
Label	Cursor label displayed at base	No

Name	Description	Read Only?
LineStyle	Style of line used for crosshair. Comma delimited string of numbers representing mark-space values. E.g. '1,1' defines short evenly spaced dots, '3,1,1,1' defines a dot-dash style	No
OldStepMethod	'On' or 'Off'. Selects method of choosing the position of the cursor when stepped to a new curve using the TAB key.	No
Style	Style of crosshair. Possible values: 'Crosshair' or 'Cursor'. 'Crosshair' means the object is displayed as a crosshair with a width and height that extends to cover the whole grid. 'Cursor' means that the object is a small bitmap representing a cross.	No
X1	X data value of crosshair position	No
Y1	Y data value of crosshair position. The value can be written but this can only affects non-monotonic curves where there are multiple y crossings at a given x value.	No

CrosshairDimension

Object used to display the dimensions and positions of cursors. There are two types, namely horizontal and vertical.

Name	Description	Read Only?
Curve1	ID of curve attached to crosshair 1	Yes
Curve2	ID of curve attached to crosshair 2	Yes
Graph	ID of parent graph	Yes
ID	ID of this object	Yes
Type	Type of object - always 'CrosshairDimension'	Yes
Vertical	0 = Horizontal dimension, 1 = Vertical dimension	Yes
XDiff	= X2-X1	Yes
YDiff	= Y2-Y1	Yes
Crosshair1	ID of crosshair 1	No
Crosshair2	ID of crosshair 2	No
Extent	Value used to define space occupied by dimension as a proportion of font size. For horizontal dimensions this is the vertical space as a proportion of font height and for vertical dimensions this is horizontal space as a proportion of average font width	No

Name	Description	Read Only?
Font	Font used to display labels. Can either be the name of a font object or a font spec as returned by GetFontSpec.	No
Hidden	Dimension is hidden	No
Label1	Label positioned to depict value of first crosshair. Default = %x1% for horizontal types, %y1% for vertical.	No
Label2	Label positioned to depict value of second crosshair. Default = %x2% for horizontal types, %y2% for vertical.	No
Label3	Label positioned to depict the separation between crosshairs. Default = %XDiff% for horizontal types, %YDiff% for vertical.	No
Position	Value defines display order of dimension. For vertical dimensions this defines the order left to right and for horizontal dimensions this defines the order bottom to top	No
Style	<p>Controls display of dimension labels. Possible values:</p> <p>Internal Show difference only (label3) - internal position</p> <p>External Show difference only (label3) - external position</p> <p>Auto Show difference only (label3), position chosen automatically</p> <p>P2P1 Show absolute labels (label1 and label2)</p> <p>P2P1Auto Show all labels. Difference position chosen automatically</p> <p>None No controls selected</p>	No
VerticalText	If set to 1, text is displayed vertically	No

Name	Description	Read Only?
X1	For horizontal types, holds the value of the x data position of the first crosshair and is readonly. For vertical types holds the x view co-ordinate location of the object on the screen and is writeable	No
Y1	For vertical types, holds the value of the y data position of the first crosshair and is readonly. For horizontal types holds the x view co-ordinate location of the object on the screen and is writeable	No
X2	For horizontal types, holds the value of the x data position of the second crosshair and is readonly. For vertical types holds the view co-ordinate location of the object on the screen and is writeable	No
Y2	For vertical types, holds the value of the y data position of the second crosshair and is readonly. For horizontal types holds the x view co-ordinate location of the object on the screen and is writeable	No

Curve

Curve objects represent all graph curves

Name	Description	Read Only?
Analysis	Analysis type used to create curve's data	Yes
DefaultLabel	This is composed from Name, Suffix and GroupName to form a text string that is the default label for the curve	Yes
DisplayType	May be: 'analog': curve is regular analog trace 'decimal': bus display showing decimal values 'hex': bus display showing hexadecimal values 'binary': bus display showing binary values	Yes
Division	Division index if plotting a multi-division vector. See Multi-division Vectors (page 38)	Yes
Expression	Expression that created this curve	Yes
Graph	ID of parent graph	Yes
GroupName	The data group that was current when the curve was created	Yes
ID	ID of this object	Yes
Limits	The X an Y limits of the curve in the form '[<i>xmin</i> , <i>xmax</i> , <i>ymin</i> , <i>ymax</i> ']'	Yes
Measurements	Measurements added to a curve	Yes

Name	Description	Read Only?
NumDivisions	Number of divisions in curves data. I.e. the number of separate traces in a group of curves. Groups of curves are usually produced by Monte Carlo analyses	Yes
Probeld	Name used to uniquely identify fixed probe (i.e. .GRAPH statement) that created this curve. Used to maintain persistence. Empty for randomly plotted curves	Yes
ShortLabel	A label composed from Name and Suffix but without group name	Yes
Type	Type of object - always 'Curve'	Yes
XAxis	ID of x-axis attached to curve	Yes
XExpression	Expression that defines X-values	Yes
XUnit	Physical type of curve's x-data	Yes
YAxis	ID of y-axis attached to curve	Yes
YUnit	Physical type of curve's y-data	Yes
Frozen	If 'true' curve will not be purged. That is it will not be removed to satisfy the persistence setting for a fixed probe	No
Label	The curve's label. This is the text that appears in the legend panel. This can use a symbolic constant and in fact defaults to %DefaultLabel%. Note that when reading back a symbolic value assigned to this property, the resolved value will be returned	No
Name	The curve's base name. This is the value passed to the /name switch of the Curve/Plot command or the name of the vector plotted if no /name switch is supplied.	No
RGBColour	Colour of curve as an RGB value	No
Sequence	Integer value that is used to define default colour	No
ShowPoints	If 'true' data point markers will be displayed	No
Suffix	This is assigned when the result of a multi-step analysis is plotted to give information about the step. E.g. if you stepped a resistor value the suffix would hold the name and value of the resistor at the step.	No
Visible	If 'false', curve will be hidden, but its legend display will remain	No

CurveMarker

An object used to title a curve or mark a feature.

Name	Description	Read Only?
Division	Division index of attached curve	Yes
Graph	ID of parent graph	Yes
ID	ID of this object	Yes
Type	Type of object - always 'CurveMarker'	Yes
Curve	ID of attached curve	No
Font	Font for label	No
Hidden	Not implemented	No
Label	Label of curve marker. May be a symbolic value. Default is %curve:label% which returns the label of the attached curve	No
LabelJustification	Text Alignment. May be one of these values: -1 Automatic 0 Left-Bottom 1 Centre-Bottom 2 Right-Bottom 12 Left-Middle 13 Centre-Middle 14 Right-Middle 8 Left-Top 9 Centre-Top 10 Right-Top	No
SnapToCurve	'On' or 'Off'. If 'On' marker tracks attached curve i.e its y position is determined by the y value of the curve at the marker's x position. If 'Off' marker may be freely located.	No
X1	X-data value at arrowhead	No
Y1	Y-data value at arrowhead	No
X2	X position of label in view units relative to arrowhead	No
Y2	Y position of label in view units relative to arrowhead	No

FreeText

Free text objects are items of text with no border or background that are not attached to any other object

Name	Description	Read Only?
Date	Date that the object was created. If the object is on a graph that has been saved to a file then subsequently restored, the date will be the date that the object was originally created.	Yes
Graph	ID of parent graph	Yes
ID	ID of this object	Yes
Parent	ID of parent object. If text is placed freely on the graph, this will be the same as the Graph property. FreeText objects, however are also used in TextBoxes in which case this returns the id for the TextBox	Yes
Time	Time that the object was created. If the object is on a graph that has been saved to a file then subsequently restored, the time will be the time that the object was originally created.	Yes
Type	Type of object - always 'FreeText'	Yes
Version	Product name and version	Yes
Font	Font for label	No
Hidden	Not implemented	No
Label	Text displayed. Symbolic values may be used. E.g. %Time% will display the time the object was created.	No
LabelJustification	As CurveMarker (see above) except -1 (automatic) not allowed	No
X1	X location of object in view units	No
Y1	Y location of object in view units	No

Graph

The Graph object represents a graph sheet

Name	Description	Read Only?
FirstCurve	ID of the oldest curve on the graph	Yes
GroupTitle	Title of the data group that was current when the graph was created	Yes
ID	ID of this object	Yes
MainCursor	ID of Crosshair object comprising the main cursor. Value = -1 if cursors are not enabled.	Yes

Name	Description	Read Only?
RefCursor	ID of Crosshair object comprising the reference cursor. Value = -1 if cursors are not enabled.	Yes
SourceGroup	The data group that was current when the graph was created	Yes
Type	Type of object - always 'Graph'	Yes
CursorStatusDisplay	Sets method of displaying cursor positions and dimensions. Possible values: Graph Display on graph using CrosshairDimension object StatusBar Display on status bar Both Display on both graph and status bar	No
Path	Path of file to save to. This is the file that will be used by the "File Save" menu. When saving a graph, this property will be set to the full path name of the file.	No
TabTitle	The text in the title of the tabbed sheet. This can be symbolic. Default is %SourceGroup% %FirstCurve:Label%	No
TitleBar	Text to be displayed in the graph window title bar when the graph's sheet is in view. This may be symbolic. Default is %SourceGroup% (%GroupTitle%)	No

LegendBox

The LegendBox is used to display labels for every curve on the graph sheet. It consists of a box that is loaded with LegendText objects - one for each curve on the graph. The LegendText objects are automatically loaded when a curve is added to the graph and automatically deleted when a curve is deleted. LegendBox is very similar to the text box and shares the same properties with the following differences and additions:

1. Type property has the value 'LegendBox'

2. LegendBox has two additional properties as shown below

Name	Description	Read Only?
Labels	Semicolon delimited string defining text entries in the box. Each entry is usually %DefaultLabel% which resolves to the value of the DefaultLabel property of the LegendText object. Other symbolic values for LegendText properties may also be used	No
LegendLabel	Text of label that is loaded into box when a curve is added to the graph. This can be symbolic in which case it references properties of the <i>LegendText</i> object that displays the text. The default value is %DefaultLabel%	No

LegendText

LegendText objects are used to load legend boxes and cannot be instantiated independently. They are similar to FreeText objects and share the same properties with the following differences and additions:

1. Type property has the value 'LegendText'
2. The Label property is set to the value of the legend box's LegendLabel property when it is added to the box.
3. LegendBox has two additional properties as shown below

Name	Description	Read Only?
Curve	ID of attached curve	Yes
DefaultLabel	The default value for the label. Actually equivalent to %Curve:Label%<n>%Curve:Measurements%. (<n> denotes a new line).	Yes

TextBox

A TextBox consists of a border with a definable background colour into which a FreeText object may be added. TextBox is also the basis of the LegendBox object.

Name	Description	Read Only?
Graph	ID of parent graph	Yes
ID	ID of this object	Yes
Type	Type of object - always 'TextBox'	Yes
AutoWidth	'On or 'Off'. If 'On' the width of the box is automatically adjusted according to its contents subject to MaxHeight	No
Colour	Background colour. Either the name of a colour object or a colour specification.	No

Name	Description	Read Only?
Font	Font used for text objects added to the box. In practice this only affects the LegendBox object which is based on TextBox.	No
Hidden	Not implemented	No
Label	Text displayed in box	No
MaxHeight	Maximum physical height in mm of box. This is only used when AutoWidth='On'	No
X1	X location of object in view units	No
Y1	Y location of object in view units	No
X2	Physical width of box in mm. (Ignored if AotoWidth='On')	No
Y2	Physical height of box in mm	No

Graph Co-ordinate Systems

Three different units of measure are used to define the location and dimensions of an object on a graph sheet. These are 'View units', 'Physical units' and 'Data units'. These are explained as follows:

'Physical Units' relate to the physical size of the displayed object and have units of millimetres. Physical units are only used for dimensions of some annotation objects and are not used for location. When objects are displayed on a screen an assumption is made for the number of pixels per inch. This depends on the display driver but is typically in the range 75 - 100.

'Data Units' relate to the units of the X and Y axes. Typically an object such as curve marker is located using data units so that it always points to the same point on a curve regardless of how the graph is zoomed or scrolled.

'View Units' relate to the current viewable area of the graph. View units use a co-ordinate system whereby the bottom left of the grid area is co-ordinate (0,0) and the top right corner of the grid is co-ordinate (1,1). View units are used to define the location of objects that need to be at a fixed location on the graph irrespective of zoom magnification.

Event Scripts

There are three special scripts that are automatically called by the SIMetrix system in response to user events. These scripts are detailed in the following table:

on_graph_anno_doubleclick	Called when the user double clicks on certain graph objects
on_accept_file_drop	Called when a file of directory is dropped on a SIMetrix window.
on_schem_double_click	Called when the left mouse button is double clicked in the schematic window.

All three scripts are defined internally but can be customised if desired. (See [“Modifying Internal Scripts” on page 435](#)). Details on these event scripts follow.

on_graph_anno_doubleclick

The script is called when some graph objects are double clicked.

The script is passed two arguments when it is called. The first is the object's ID and the second is specific to the object that is double clicked. Currently the second argument is only used by curves and is set to its division index.

on_accept_file_drop

This is called when an a file, folder or group or files and/or folders is dropped on the command shell or a schematic or graph window.

Two arguments are passed. The first identifies the window type. This may be one of:

Schematic	Schematic window
Graph	Graph window
Shell	Command shell

The second argument contains a list of full path names of the objects dropped. The items are separated by semi-colons.

on_schem_double_click

Script is called when the left mouse button is double clicked in the schematic window. Two arguments are supplied providing the x and y coordinates of the mouse at the time the double click event occurred.

IMPORTANT: This script is only called if the schematic double click mode is set to ‘Edit Selected Component’. See options dialog box (menu File|Options|General...). In ‘Classic’ mode it is not called at all.

User Defined Script Based Functions

Overview

The SIMetrix script language provides a method of creating user defined functions that can be used in any front end expression. These expressions may be used in scripts, on the command line and even within a schematic template property.

User defined functions are used to define some of the goal functions designed for performance and histogram analysis. The scripts for these all begin “uf_” and are registered using the “reg_user_funcs” script. The source for these can be found on the installation CD.

Defining the Function

User defined functions are defined as a script. The arguments to the function and the return value from the function are passed as the script's arguments. The script's first argument is passed by reference and is the return value while the remaining arguments are the arguments passed in the call to the function. The function may have up to seven arguments and they may be of any type. See example below.

Registering the Script

For the expression evaluator to recognise the function name, the script and function name must be registered. This is done with the RegisterUserFunction command. The definition of this is:

```
RegisterUserFunction Function-Name Script-Name [min-number-args]  
[max-number-args]
```

For details see [page 408](#).

Note that function registration is not *persistent*. That is the registration only lasts for the current session. If you wish to make a permanent function definition, place the RegisterUserFunction command in the startup script.

Example

Here is a trivial example. The following shows the steps to create a function that multiplies a number by 2. First the script

```
Arguments @rv arg1  
Let rv = 2*arg1
```

Save this to a file called - say - times_two.sxscl and place it in the script directory.

Now, register the script as a function called "Times2". To do this, execute the command:

```
RegisterUserFunction Times2 times_two 1 1
```

The definition is now complete. To test it type at the command line:

```
Show Times2(2)
```

You should see the result:

```
Times2(2) = 4
```

User Defined Binary Functions

Overview

From version 5, it is possible to develop script functions written in 'C' or 'C++' and compile them into a DLL/shared library to be loaded into SIMetrix as a plugin.

This makes it possible to perform complex processing on data that would run too slowly using the interpreted script language.

Documentation

Documentation and associated header and example files are provided on the install CD. See the directory CD/Script/user-function-interface.

Non-interactive and Customised Printing

Overview

The SIMetrix script language provides a number of functions and commands that allow *non-interactive printing*. That is printing without user intervention. This is useful for - say - running multiple simulations in the background and automatically printing the results when the simulation is complete. The same printing facilities may also be used to customise the layout of printed schematics and graphs. The user interface provides a method of printing a single graph and schematic on the same sheet, but other arrangements are possible using the underlying printing commands.

The available printing commands are:

ClosePrinter [page 350](#)
NewPrinterPage [page 392](#)
OpenPrinter [page 395](#)
PrintGraph [page 401](#)
PrintSchematic [page 401](#)

The functions are

GenPrintDialog [page 141](#) (for interactive printing)
GetPrinterInfo [page 185](#)

Each of these commands and functions is described in detail in its relevant section. Here we give a general overview for the printing procedure.

Procedure

The sequence for a print job is:

1. Open printer. At this stage the printer to be used, page orientation, title and number of copies may be selected.
2. Print pages. The actual graphs/schematics to be printed along with scaling and margins are specified here. Any number of pages can be printed.
3. Close printer. This actually starts the physical printing. It is also possible to abort the print job.

Example

Suppose we wish to create a PDF file using 'Acrobat Distiller' for the current graph. Of course you can readily do this by selecting File|Print... and making the appropriate

selections using the dialog box. This is no good, however, if you want to create a PDF file for a graph created using an automated simulation, perhaps run overnight. The following script will do this without user intervention.

```
** Get info on system printers
Let printInfo = GetPrinterInfo()

** Search for acrobat distiller. The printer list from Get-PrinterInfo
** starts at index 2 so we subtract 2 to get the index
** needed by OpenPrinter
Let distillerIndex = Search(printInfo, 'Acrobat Distiller')-2

** If Acrobat distiller is not on the system
** Search will return -1
if distillerIndex<0 then
    Echo "Acrobat Distiller is not installed"
    exit script
endif

** Open Printer using distiller.
** Orientation will be landscape which is the default
** Number of copies = 1.
** The title will be used by distiller to compose the file name
** for the PDF file i.e. Graph1.PDF
OpenPrinter /title Graph1 /index {distillerIndex}

** Now print the graph
** Major axis on minor axis off. All margins 20mm.
PrintGraph /major on /minor off /margin 20 20 20 20 /caption "Test Print"

** Close Printer. This will actually start the print
ClosePrinter
```

You can of course replace 'Acrobat Distiller' with any printer that is on your system. You must use the printer's name as listed in the Printers section of the system control panel. You can also find a list of system printers from within SIMetrix by typing at the command line:

```
Show GetPrinterInfo()
```

The first two values are numbers but the remaining are the currently installed printers on your system.

If you omit /index switch for the OpenPrinter command, the application default printer (not the system default printer) will be used. The application default printer is the same as the system default printer when SIMetrix starts but will change whenever the user selects a different printer using the SIMetrix File|Print... dialog box.

Schematic Template Scripts

Overview

Schematic template scripts are a method of performing advanced netlist processing. The TEMPLATE property can be used to customise the netlist entry for a device and it

has a number of features that allow quite complex devices to be created. However, the template syntax is not as powerful as a full featured programming language and this makes more complex devices very difficult to implement.

To overcome this the template script feature was developed. With this method a script is called during the netlist generation phase for every instance that possesses a `TEMPLATESCRIPT` property. A script can then generate the netlist entry for that instance. With this method there is no limit to the complexity of generated devices.

Defining a Symbol for a Template Script

To use the template script feature, the schematic symbol must specify the script to be called. This is done quite simply by adding a property with the name `TEMPLATESCRIPT` and giving it a value that defines the path of the script. If a full path isn't given (and we recommend that you don't use a full path), SIMetrix will search the directory where the netlist resides followed by the `SCRIPT` directory for the specified file. If the file is not found, no error message will be output and the device netlist line will be created as if no template script was defined.

When is the Template Script Called?

The template script is called for each instance just before its netlist entry is generated. The `REF` property of the instance is passed to the script as an argument along with the name of the property used for the template. The script controls the netlist output by calling the `TemplateSetValue` (page 428) command.

The Template Script

The script is passed two string arguments. These are:

1. The value of the `REF` property of the instance being processed.
2. The name of the template property being used for that instance. This is usually `'TEMPLATE'` but for `SIMPLIS` netlists it is usually `'SIMPLIS_TEMPLATE'`. There is also a netlist option to change the name of the template property.

There are two functions and two commands that are designed specifically for template scripts and indeed they cannot be used anywhere else. The commands and functions are listed below.

The most important command is `TemplateSetValue`. This is what you must use to define the netlist entry. The value supplied to this command defines the template that will be used to create the netlist entry. It can of course provide a completely literal netlist line, but more usually some template keywords would be used.

Template Commands and Functions

This is a brief summary. See the entries in the reference pages for more details.

TemplateResolve(ref, template)

Performs the same process that is usually done on a template property except that it uses the template that you supply as an argument not the device's template. *ref* is the `REF` property of device being processed.

TemplateGetPropValue(ref, prop)

Returns the value of the property *prop*. You should use this function not PropValues() to get at property values. It is faster than PropValues() but won't work in regular scripts.

TemplateEditProperty ref proname propvalue

Edits a property's value. Like TemplateGetPropValue it is much faster than the regular commands but only works in a template script.

TemplateSetValue ref templatevalue

Changes the value of the template used to create the netlist line currently being compiled. Does *not* change the template property itself.

Creating and Modifying Toolbars

From version 5, SIMetrix allows the complete customisation of toolbars. You can modify the definitions of existing toolbars and buttons, as well as create new toolbars and new tool buttons. This section explains how.

Modifying Existing Toolbars and Buttons

You can rearrange the button layout of existing toolbars by modifying the 'Set' option variables that define them. In the case of the schematic component buttons, this can be done via a simple GUI. See menu View|Configure Toolbar... .

For other toolbars use the command [Set \(page 420\)](#) to reassign the buttons. The following table shows the name of the 'Set' variable to use for each one.

'Set' Variable Name	Toolbar
ComponentButtons	SchematicComponents (non 'Micron' versions)
MicronComponentButtons	SchematicComponents ('Micron' versions)
CommandShellMainButtons	CommandShellMain
CommandShellMainNoSchemButtons	CommandShellMain (if schematic disabled - OEM versions only)
SIMPLISComponentButtons	SIMPLISComponents
SchematicMainButtons	SchematicMain
SchematicFileButtons	SchematicFile
GraphMainButtons	GraphMain

The 'Set' variable should be set to a value consisting of a semi-colon delimited list of valid button names. For a list of pre-defined buttons, see [page 469](#).

For example, the following will add a 'New Schematic' button to the schematic file tool bar:

```
Set SchematicFileButtons="SchemNew;SchemOpen;SchemClose;SchemSave"
```

You can also use 'Unset' to restore a toolbar to its default setting. E.g.

```
Unset SchematicFileButtons
```

will restore the schematic file toolbar to just three buttons without the new schematic button.

To determine the current definition, use the GetOption function with the 'Set' variable name as described in the table above. For example

```
Show GetOption('SchematicFileButtons')
```

will display in the message window the current definition for the SchematicFile toolbar.

Redefining Button Commands

You can change the command executed when a button is pressed using the command [DefButton \(page 361\)](#). This is useful if you want to change the symbol placed for one of the component buttons. For example if you wanted to change one of the NMOS buttons, you could do something like:

```
DefButton NMOS4 "inst /ne my_nmos"
```

redefines the four-terminal NMOS button to place a symbol with name my_nmos.

You can redefine any of the pre-defined buttons. See [page 469](#) for a complete list.

Defining New Buttons and Editing Buttons

You can define completely new buttons with your own graphic design and add them to an existing toolbar. The same method can also be used to redefine the graphics for existing buttons.

This is done using the command [CreateToolButton \(page 356\)](#). These are the steps to take:

1. Create a graphical image for the button. This should be in a windows bitmap (.bmp), portable network graphic (.png) or JPEG (.jpg) format. You can use almost any paint application to do this. But, if you want to define a mask - that is you wish to define transparent areas - then you must use an editor capable of creating 'portable network graphics' (PNG) images. However, this is rarely necessary in practice and none of the built in graphics define a mask. This is because SIMetrix will automatically create one that makes the area outside the perimeter of the image transparent. The result is usually satisfactory.

You can make your graphic any size, but to be compatible with the built-in images, you should make them 16x16 pixels. The built-in graphics are all 16

colour, but you can use any colour depth supported by your system.

When you have created your image, you should save or copy it to the images directory. This is located at:

Windows: `simetrix-root\support\images`

Linux: `simetrix-root/share/images`

where `simetrix-root` is the top level directory in the SIMetrix tree.

2. Execute the command [CreateToolButton \(page 356\)](#). As with menu and key definitions, the definitions created by this command are not *persistent* that is they will be lost when SIMetrix exits. To make permanent definitions, you should place the commands in the start up script. See [Startup Script \(page 47\)](#) for more details.

CreateToolButton will not add the button to any toolbar nor does it assign a command to be executed when it is pressed. These operations are described in the following steps

3. Define a command to be executed when this button is pressed. This is done using the command [DefButton \(page 361\)](#). Again, this should be place in your startup script.
4. Add the button to a toolbar. See “[Modifying Existing Toolbars and Buttons](#)” on [page 466](#) to find out how to add this to an existing toolbar. If you wish to create a new toolbar for the new button, see “[Creating New Toolbars](#)” below.

For example, suppose you created a symbol for a diffused resistor and wanted to assign this to a toolbar button that is distinct from the regular resistor button. These are the steps:

1. First you would create a graphical image called, for example, `diffres.png`. Copy this to the images directory as described above.
2. Execute (or place in startup script):

```
CreateToolButton /class component diffres diffres.png "Place  
Diffused Resistor"
```

(This must all be on one line)

This will create a button called ‘diffres’ that we will refer to in the following steps. The switch ‘class component’ identifies the button as one that places a component and so will be listed in the GUI based system to edit component toolbars. (See schematic menu View|Configure Toolbar...). This will make adding the button to a component toolbar a simple operation.

3. Execute (or place in startup script):

```
DefButton diffres "inst /ne diffressym"
```

where `diffressym` is the name of the schematic symbol created for the diffused resistor.

4. To add to the button to a component toolbar, simply select schematic menu View|Configure Toolbar... You should see ‘Place Diffused Resistor’ on the left

hand side. Select and press Add to add to the toolbar, then use the up down buttons to choose a suitable position.

It's a little harder to edit non-component toolbars as there is currently no GUI to perform the operation in step 4 above. For pre-defined toolbars you can obtain the current specification using the `GetOption` function and then add your new button to the resulting value at an appropriate location. Then use the `Set` command to redefine the toolbar. See “[Modifying Existing Toolbars and Buttons](#)” on page 466 for more details.

Creating New Toolbars

To create a completely new toolbar, use the command `CreateToolBar` (page 355). This will create an empty toolbar.

To add buttons to a new toolbar, you must use the command `DefineToolBar` (page 362). You can add both pre-defined and user-defined buttons to a custom toolbar.

Pre-defined Buttons

The following table lists all the buttons that are pre-defined. All of these buttons may be redefined if required.

The bitmaps are embedded in the SIMetrix binary, but can also be found on the install CD in the directory `script/images`.

The command executed by each button can be found using the command `ListStdButtonDefs` (page 386).

Button name	Description	Bitmap
AddCurve	Add Curve	newcurve.bmp
AddFourier	Fourier...	newfourier.bmp
BiasV	Place Bias Marker	biasv.bmp
CalcAveragePower	Display Average Power/Cycle	avg.bmp
CalcFall	Display Fall Time	falltime.bmp
CalcHighPass3db	Display -3dB Point (High Pass)	3dbhighpass.bmp
CalcLowPass3db	Display -3dB Point (Low Pass)	3dblowpass.bmp
CalcRise	Display Rise Time	risetime.bmp
CalcRMS	Display RMS/Cycle	rms.bmp
Capacitor	Place Capacitor	cap.bmp
Copy	Duplicate	copy.bmp
Delete	Cut	erase.bmp

Button name	Description	Bitmap
DeleteAxis	Delete Axis/Grid	delgrid.bmp
DeleteCurve	Delete Curve	delete.bmp
Diode	Place Diode	diode.bmp
Flip	Flip	flip.bmp
GraphClose	Close Graph	fileclose.bmp
GraphOpen	Open Graph	fileopen.bmp
GraphSave	Save Graph	filesave.bmp
Ground	Place Ground	gnd.bmp
HideCurves	Hide Selected Curves	hide.bmp
IGBT	Place IGBT	igbt.bmp
Inductor	Place Inductor	ind.bmp
IProbe	Place Current Probe	iprobe.bmp
ISource	Place Current Source	isource.bmp
Mirror	Mirror	mirror.bmp
MoveCurve	Move Curve to Selected Axis/Grid	movecurve.bmp
NewAxis	New Axis	newaxis.bmp
NewGrid	New Grid	newgrid.bmp
NJFET	Place N-channel JFET	njfet.bmp
NMOS	Place N-channel MOSFET	nmos.bmp
NMOS3IC	Place 3 term N-channel MOSFET	nmos_ic3.bmp
NMOS4	Place 4 term N-channel MOSFET	nmos_ic.bmp
NPN	Place NPN Transistor	npn.bmp
Opamp	Place Opamp	opamp.bmp
Options	Options	options.bmp
PJFET	Place P-channel JFET	pjfet.bmp
PMOS	Place P-channel MOSFET	pmos.bmp
PMOS3IC	Place 3 term P-channel MOSFET	pmos_ic3.bmp
PMOS4	Place 4 term P-channel MOSFET	pmos_ic.bmp

Button name	Description	Bitmap
PNP	Place PNP Transistor	pnp.bmp
Print	Print	print.bmp
PSU	Place PSU	psu.bmp
Resistor	Place Resistor (Box shape)	res.bmp
ResistorZ	Place Resistor (Z shape)	resz.bmp
Rotate	Rotate	rotate.bmp
SatInd	Place Saturable Inductor	sat_ind.bmp
SatTx	Place Saturable Transformer	tx_sat.bmp
SchemClose	Close Schematic	fileclose.bmp
SchemNew	New Schematic	newschem.bmp
SchemOpen	Open Schematic	fileopen.bmp
SchemSave	Save Schematic	filesave.bmp
SchemSaveAll	Save All Schematics	saveall.bmp
SCR	Place Thyristor	scr.bmp
ShowCurves	Show Selected Curves	show.bmp
SimPause	Pause Simulation	pause.bmp
SimRunNetlist	Run Netlist	run.bmp
SimRunSchem	Run Schematic	run.bmp
SymbolNew	New Symbol	newsymbol.bmp
TitleCurve	Change Curve Name	curvetitle.bmp
TL	Place Transmission Line	tl.bmp
Tx	Place Transformer	tx.bmp
Undo	Undo	undo.bmp
UndoZoom	Undo Zoom	undo.bmp
VProbe	Place Voltage Probe	vprobe.bmp
VSource	Place Voltage Source	vsource.bmp
Waveform	Place Waveform Generator	vsig.bmp
Wire	Wire Mode	pencil.bmp
Zener	Place Zener Diode	zener.bmp

Button name	Description	Bitmap
ZoomFull	Fit Window	zoomfull.bmp
ZoomIn	Zoom In	zoomin.bmp
ZoomOut	Zoom Out	zoomout.bmp
ZoomRect	Zoom Box	zoomrect.bmp
ZoomXAuto	Fit Width	zoomwidth.bmp
ZoomYAuto	Fit Height	zoomheight.bmp

Custom Dialog Boxes

Overview

SIMetrix has a feature that permits the creation of custom dialog boxes without the need to write program code. This can be done using a special graphical tool called the “SIMetrix Dialog Designer” supplied with SIMetrix from version 5.3. SIMetrix Dialog Designer is derived from a commercial tool developed by Trolltech AS who supply us with the Qt library used for SIMetrix UI development. Trolltech have kindly given us permission to ship this tool with SIMetrix. Note that “SIMetrix Dialog Designer” is a stripped down version of the full commercial product.

Currently we supply only a Windows version of this tool, but the dialogs generated will work with Linux versions of SIMetrix.

Starting “SIMetrix Dialog Designer”

The tool is installed with the rest of the SIMetrix binaries and is called “designer.exe”. Use windows explorer to locate designer.exe in the “bin” folder under the SIMetrix root. The SIMetrix installer does not create a short cut to this but you may create one yourself if required.

Developing Dialogs

The basic procedure is:

1. Start Designer
2. Select “Dialog” under “New File/Project”
3. Set the form’s name property to the required name of the SIMetrix function.
4. Edit caption property as required
5. Add widgets as required. See next section for further details. See also “Using Geometry Management” on page 4
6. Save result as an .sxdlg file to the directory support\dialogs under SIMetrix root (Windows) or /usr/local/share/dialogs (Linux). This is the default location for user dialogs. There is an option setting that allows them to be located elsewhere. See below for details.

The dialog is now designed. If SIMetrix is currently running, shut it down and restart it to register the new dialog function.

Note that you do not need to restart after editing the dialog - only when creating it for the first time or when changing the function name. SIMetrix registers the filename and function name on startup, but will reread it when the function is called. This means that you can make changes to your dialog without having to shut down and restart SIMetrix each time.

You can select a different location for user dialogs with the option setting `UserDialogsDir`. Type this at the command line:

```
Set UserDialogsDir=path
```

where `path` is the full path of the new dialogs location. You may use logical path symbols in the definition. For example `“%SXAPPDAPATH%/userdialogs”` resolves to a directory under the application data path. Note that you must restart SIMetrix after changing the path.

The Widgets

“Widgets” are the dialog elements such as edit boxes and push buttons that you use to enter data and choices. In Windows “Widgets” are sometimes called “Controls”.

A range of special widgets is supplied that have some extra properties to define how they will be initialised when the dialog is opened and what they will return through the SIMetrix script function call mechanism. These widgets can be found under the “SIMetrix” group. Always use these for anything used for data entry. Other widgets that do not require initialisation nor output data may also be used. E.g. the items under “Containers”. Note that the “Radio Button” widget in the “Buttons” group can only be used inside a “RadioGroup” which you will find in the SIMetrix group.

In general data is transferred to the dialog widgets by the arguments of the SIMetrix script function. Each argument is an array of strings and each widget may specify through its properties the argument index and the array element index where the data is located. In every case the data is a single string. If multiple values are required for a widget, it will either have multiple properties to define them, or, in the case of lists of values, the items will be delimited by a pipe (‘|’) symbol.

Data is returned in a similar manner. But as there is only one return value, just a single array element is specified.

General Properties

There are five user settable properties in use by the various widgets, but not all widgets use all of the properties. Some widgets may have additional special purpose properties. These five general properties are:

Property Name	Description
argIndex	Index of script function argument used for initialisation of widget. First argument has index=0. You may use a maximum of 8 arguments so this property may not be larger than 7
inElementIndex	Index into array supplied to argIndex for value used to initialise widget. First element has index=0
outElementIndex	Index into array returned by script function for user entered value
itemsArgIndex	Index of script function argument used to supply items to initialise list. Items separated by pipe (' ') symbol. Currently used by list boxes and combo boxes
itemsElementIndex	Index into array supplied to itemsArgIndex for items to initialise list. Items separated by pipe (' ') symbol. Currently used by list boxes and combo boxes

Full details and examples for each widget type follow:

EditBox

The properties argIndex, inElementIndex and outElementIndex initialise and return the text value stored in a single line edit box.

TextEdit

As EditBox but multi-line.

Spinner

Used for entering numeric values. argIndex, inElementIndex and outElementIndex used to initialise and return. Note that box stores a numeric value, but the script arguments must still be strings.

This widget has the following properties that govern its behaviour:

Property name	Description
engMode	Boolean. If true, value is always displayed in engineering notation using suffixes such as m, u, k etc
step125	Boolean. If true, spinner buttons step in 1-2-5 sequence. Otherwise they step in a linear sequence controlled by the 'increment' property
increment	Increment for spinner buttons. Only effective if 'step125' property is false
max	Maximum value allowed for widget
min	Minimum value allowed for widget
precision	Value displayed and returned to precision specified.
allowExpressions	If true, the user may enter expressions enclosed with curly braces: '{' and '}'. If false, only numeric values will be allowed

CheckBox

A check box providing a simple on-off selection. `argIndex`, `inElementIndex` and `outElementIndex` used to define initial setting and return value in normal way. '1' indicates checked and '0' indicates unchecked.

Label

Static label. Can be set with static value in which case `argIndex` and `inElementIndex` should be -1. Alternatively can be initialised via function call using `argIndex` and `inElementIndex`. Does not return a value.

RadioGroup

A container that should be filled with one or more Radio Buttons (these may be found under the "Buttons" group). Only one of the radio buttons in the group may be checked at any time. The usual properties are used to initialise and the return values. '0' means check the top most button, '1' the second button, '2' the third etc.

PushButton

A push button with two alternative modes of operation. If the property 'toggleButton' is false, then this may be used to close the dialog box. In this case the property 'action' must be set to either 'reject' or 'accept'. If 'reject' is set then the dialog box function will return an 'empty vector'. That is the array returned will have a length of zero. (You must test this with the script language's `length()` function). If set to 'accept' the normal data will be returned. The 'outElementIndex' property may be set in this case in which case the value returned will be

If 'toggleButton' is set to true then 'action' must be set to 'none' to be meaningful. In this case the button will toggle on or off. The return value controlled by

outElementIndex will be either 'on' or 'off'. Currently there is no method to initialise the toggle state. This will be corrected in a later release.

CancelButton and OkButton

These are identical to PushButton except for changes to default values of some properties. "Cancel Button" behaves as a button to cancel a dialog and will cause the calling function to return an empty vector. "Ok Button" closes a dialog and accepts the user's input.

ListBox

A list box containing a list of values. The values themselves are defined using itemsArgIndex and itemsElementIndex properties and must be in the form of a single string containing a list of values separated by a pipe symbol.

The initial value selected is defined by argIndex and inElementIndex. This is the actual value not the index into the list. The item selected in the list is returned in outElementIndex.

ComboBox

A drop down "combo box" otherwise the same as the ListBox.

ParameterView

This is experimental and currently unsupported.

Using Geometry Management

SIMetrix Dialog Designer features an advanced system, known as geometry management, that automatically arranges widgets in the dialog. Geometry management controls the position and size of the widgets in a manner that maintains the layout in an aesthetically pleasing form even if the dialog is resized.

These features are available via the "Layout" menu, via the toolbar and also with the context popup menu. The features available are:

1. Layout horizontally. Lays out selected widgets in a horizontal line
2. Layout vertically. Lays out selected widgets in a vertical line
3. Layout in a grid. Lays out widgets in a grid arrangement using their initial position as a guide
4. Layout vertically/horizontally in a splitter. Lays out two widgets with a splitter bar in between allowing the user to control their relative sizes

The geometry management actions work on either selected widgets or all the widgets in a selected container. If no widget or container is selected, the action will be applied to all the widgets in the form. A container is a widget that is designed to hold other widgets. The containers are the widgets in the containers group and also the RadioGroup widget in the SIMetrix group.

The best way to learn about geometry management is to experiment with various widgets and containers. You may need to use the “spacer” widget available from the toolbar to provide empty spaces. Some widgets (e.g. buttons) resize to fill the space available and this is not always desirable.

Further documentation on the Designer tool can be found at the developer’s web site:

<http://doc.trolltech.com>. The version currently in use is 3.3. See under “Tools”, the SIMetrix dialog designer is based on “Qt Designer”.

Examples

A number of trivial examples are supplied that demonstrate each of the widgets. These are supplied in the examples directory under `scripts/dialogs`. To use them you must copy them to the `support/dialogs` folder (Windows) or `/usr/local/share/dialogs` folder (Linux). Here is a list:

EditDialog

Simple dialog with an edit box and an Ok button. Type:

```
Show EditDialog('Initial message')
```

to see what it does.

TestCombo

Demo of combo box, try this:

```
Show TestCombo('bill', 'fred|bill|john')
```

TestFunction

A spinner and a check box. Try:

```
Show TestFunction(['2.345', '1'])
```

ListBoxFunction

A list box and a check box, Try this

```
Show listboxfunction(['john','l'], 'fred|bill|john')
```

TextEditTest

TextEdit and two push buttons, one of them with toggle action.

Try this:

```
Show textedittest('A message')
```

JohnsModelDialog

Bits and pieces. Try this:

```
Show johnsmodeldialog(['bill', '2.345', '4.567', '1'],  
 'fred|bill|john')
```

RadioTest

A couple of radio buttons and a toggle button

```
Show radiotest('1')
```

ExecuteDialog Function

The ExecuteDialog function executes a .sxdlg file directly using the dialog definition's full path name. The first argument to this function is the full path to the dialog .sxdlg file and subsequent arguments are the dialog's arguments shifted one place. So argument 0 of the dialog function is argument 1 of ExecuteDialog. Note that the first argument must be a full path, but you may use logical path symbols.

ExecuteDialog does not require the .sxdlg file to present when SIMetrix starts up unlike the usual method of calling the dialog functions.

All script functions are limited to a maximum of 8 arguments and ExecuteDialog is not an exception. Because the first argument is reserved for the path name, this means that the maximum number of arguments that can be passed to the dialog is 7. If calling the dialog directly, the limit is 8.

Performance

Complex dialog designs can take a noticeable time to open. This is because the definition file is read and parsed every time the dialog function is called.

Copyright © SIMetrix Technologies Ltd. 1992-2008
SIMetrix 5.5 Script Reference Manual