

SIMetrix / SIMPLUS

SCRIPT REFERENCE MANUAL

VERSION 9.0

NOVEMBER 2021

SIMETRIX/SIMPLIS SCRIPT REFERENCE MANAUL

COPYRIGHT © SIMETRIX TECHNOLOGIES LTD. 1992-2021

COPYRIGHT © SIMPLIS TECHNOLOGIES INC. 1992-2021

SIMetrix Technologies Ltd.,
78 Chapel Street,
Thatcham,
Berkshire
RG18 4QN
United Kingdom

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

SIMPLIS Technologies, Inc.
P.O. Box 40084
Portland, OR 97240-0084
USA

Tel: +1 503 766 3928
Fax: +1 503 296 5674
Email: info@simplistechnologies.com
Web: <http://simplistechnologies.com>



Contents

1	Introduction	1
2	The SIMetrix Script Language	2
2.1	A Tutorial	2
2.1.1	Example 1: Hello World!	2
2.1.2	Example 2: An Introduction to Loops	2
2.1.3	Example 3: Cross Probing	3
2.1.4	Example 4: Making a Parts List	4
2.2	Variables, Constants and Types	6
2.2.1	Variable names	6
2.2.2	Types	7
2.2.3	Constants	7
2.2.4	Creating and Assigning Variables	8
2.2.5	Special Characters	8
2.2.6	Vectors	9
2.2.7	Scope of Variables, Global Variables	9
2.2.8	Empty Values	10
2.2.9	Empty Strings	10
2.2.10	Quotes: Single and Double	10
2.3	Expressions	11
2.3.1	Operators	11
2.3.2	Functions	12
2.3.3	Braced Substitutions	13
2.3.4	Bracketed Lists	13
2.3.5	Type Conversion	13
2.3.6	Aliases	14
2.4	Statements and Commands	14
2.4.1	Commands	14
2.4.2	Command Switches	14
2.4.3	If Statement	15
2.4.4	While Statement	15
2.4.5	For Statement	16
2.4.6	Foreach Statement	16
2.4.7	Foreachdiv Statement	17
2.4.8	Script Statement	17
2.4.9	Exit Statement	17
2.5	Accessing Simulation Data	17
2.5.1	Overview	17
2.5.2	Groups	18
2.5.3	Multi-division Vectors	19
2.6	User Interface to Scripts	22
2.6.1	Dialog Boxes	22
2.6.2	User Control of Execution	22

2.7	Errors	23
2.7.1	Syntax Errors	23
2.7.2	Execution Errors	23
2.8	Executing Scripts	23
2.8.1	Script Arguments	24
2.8.2	Built-in Scripts	25
2.8.3	Debugging Scripts	25
2.8.4	Startup Script	26
2.9	Unsupported Functions and Commands	26
3	Function Summary	27
3.1	Functions by Application	55
3.1.1	Configuration/Licensing	55
3.1.2	Data fitting	56
3.1.3	Dialogs	56
3.1.4	File/Directory	57
3.1.5	Graph	57
3.1.6	Mathematical	58
3.1.7	Miscellaneous	59
3.1.8	Model Library	59
3.1.9	Monte Carlo Distribution	59
3.1.10	Monte Carlo/Sensitivity/Worst-case support	59
3.1.11	Optimiser	59
3.1.12	SIMPLIS	60
3.1.13	Schematic	60
3.1.14	Schematic Styles	61
3.1.15	Schematic Symbols and Library	61
3.1.16	Script	61
3.1.17	Simulator	61
3.1.18	String	62
3.1.19	System	62
3.1.20	Text Editor	63
3.1.21	User Interface	63
3.1.22	Vector	64
3.1.23	Vectors/Groups	64
3.1.24	XML Processing	64
4	Function Reference	65
4.1	abs	65
4.2	acos	65
4.3	acosh	65
4.4	ACSourceDialog	66
4.5	ACSourceDialogStr	66
4.6	AddConfigCollection	67
4.7	AddGraphCrossHair	67
4.8	AddModelFiles	68
4.9	AddPropertyDialog	69
4.10	AddRemoveDialog	69
4.11	AddRemoveDialogNew	70
4.12	AddSymbolFiles	71
4.13	AppendSensitivityData	71
4.14	area	72
4.15	arg	73
4.16	arg_rad	73
4.17	Ascii	74
4.18	asin	74

4.19	asinh	74
4.20	AssociateModel	74
4.21	atan	75
4.22	atan2	75
4.23	atan2_deg	76
4.24	atan_deg	77
4.25	atanh	77
4.26	avg	77
4.27	BoolSelect	78
4.28	Branch	79
4.29	BuildMclogHTML	79
4.30	BuildParameterString	80
4.31	BuildSensitivityHTML	81
4.32	BuildWorstCaseHTML	82
4.33	CanOpenFile	82
4.34	ChangeDir	83
4.35	Char	83
4.36	CheckLaplaceExpression	84
4.37	ChooseDir	84
4.38	ChooseDirectory	85
4.39	Chr	85
4.40	CloseEchoFile	86
4.41	CloseFile	86
4.42	CloseSchematic	86
4.43	CloseSchematicTab	87
4.44	Coll	87
4.45	CollateVectors	87
4.46	CommandStatus	89
4.47	CompareSymbols	89
4.48	ComposeDigital	90
4.49	ConvertFromBase64	91
4.50	ConvertHTMLcolourToRGB	92
4.51	ConvertIsoTimeToUnix	92
4.52	ConvertLocalToUnix	93
4.53	ConvertNumberFromBase64	93
4.54	ConvertNumberToBase64	94
4.55	ConvertRGBcolourToHTML	94
4.56	ConvertToBase64	95
4.57	ConvertUnixTimeToIso	95
4.58	ConvertUnixToLocal	96
4.59	CopyDivisionData	96
4.60	CopyTree	97
4.61	CopyURL	97
4.62	cos	98
4.63	cos_deg	99
4.64	cosh	99
4.65	CountChars	99
4.66	CreateDiodeDialog	100
4.67	CreateGraphMeasurement	100
4.68	CreateLockFile	101
4.69	CreateNewTitleBlockDialog	102
4.70	CreateSharedAxisConnector	103
4.71	CreateShortcut	103
4.72	CreateTimer	104
4.73	CurveEditDialog	105
4.74	CurveFit	105

4.75	CurveFitLog	106
4.76	CurveFitLogX	106
4.77	CurveFitLogY	107
4.78	cv	108
4.79	CyclePeriod	108
4.80	Date	109
4.81	db	109
4.82	DCSourceDialog	110
4.83	DCSourceDialogStr	110
4.84	DefineADCDialog	110
4.85	DefineArbSourceDialog	111
4.86	DefineBusPlotDialog	112
4.87	DefineCounterDialog	113
4.88	DefineCurveDialog	113
4.89	DefineDACDialog	115
4.90	DefineFourierDialog	116
4.91	DefineFourierProbeDialog	117
4.92	DefineIdealTxDialog	120
4.93	DefineLaplaceDialog	122
4.94	DefineLogicGateDialog	123
4.95	DefinePerfAnalysisDialog	124
4.96	DefineRegisterDialog	124
4.97	DefineRipperDialog	125
4.98	DefineSaturableTxDialog	125
4.99	DefineShiftRegDialog	127
4.100	DefineSimplisMultiStepDialog	127
4.101	DeleteConfigCollection	128
4.102	DeleteTimer	128
4.103	DeleteTouchstone	129
4.104	DeleteTree	129
4.105	DeleteTreeProgress	130
4.106	DelSchemProp	131
4.107	DescendDirectories	131
4.108	DescendHierarchy	132
4.109	DialogDesigner	132
4.110	diff	133
4.111	DirectoryIsWriteable	133
4.112	Distribution	134
4.113	EditArcDialog	136
4.114	EditAxisDialog	136
4.115	EditBodePlotProbeDialog	138
4.116	EditBodePlotProbeDialog2	138
4.117	EditCrosshairDimensionDialog	139
4.118	EditCurveMarkerDialog	140
4.119	EditDeviceDialog	141
4.120	EditDigInitDialog	143
4.121	EditFileDefinedPWLDialog	143
4.122	EditFreeTextDialog	146
4.123	EditGraphMeasurement	146
4.124	EditGraphTextBoxDialog	147
4.125	EditJumperDialog	147
4.126	EditLegendBoxDialog	148
4.127	EditObjectPropertiesDialog	148
4.128	EditPinDialog	150
4.129	EditPotDialog	151
4.130	EditProbeDialog	151

4.131	EditPropertyDialog	154
4.132	EditReactiveDialog	154
4.133	EditSelect	156
4.134	EditSimplisLaplaceFilterDialog	157
4.135	EditSimplisMosfetDriverDialog	157
4.136	EditStylesDialog	159
4.137	EditSymbolBusDialog	160
4.138	EditTimer	161
4.139	EditWaveformDialog	162
4.140	EditWaveformStrDialog	163
4.141	ElementProps	165
4.142	EncodeImageToBase64	166
4.143	EnterTextDialog	167
4.144	EpochTime	167
4.145	erf	168
4.146	erfc	168
4.147	EscapeString	168
4.148	EscapeStringEncode	169
4.149	ev	170
4.150	Execute	171
4.151	ExistCommand	171
4.152	ExistDir	172
4.153	ExistFile	172
4.154	ExistFunction	172
4.155	ExistSymbol	173
4.156	ExistVec	174
4.157	exp	174
4.158	fd	174
4.159	fft	175
4.160	Field	175
4.161	FileToString	176
4.162	FilterEditMenu	177
4.163	FilterFile	177
4.164	FinalDivision	178
4.165	FindGraphMeasurement	178
4.166	FindModel	179
4.167	FIR	179
4.168	Floor	180
4.169	floorv	181
4.170	FormatNumber	181
4.171	Fourier	182
4.172	FourierOptionsDialog	182
4.173	FourierWindow	183
4.174	FullPath	183
4.175	gamma	184
4.176	Gauss	184
4.177	GaussLim	185
4.178	GaussTrunc	186
4.179	GenPrintDialog	186
4.180	GetActualPath	188
4.181	GetAllAxes	188
4.182	GetAllCurves	188
4.183	GetAllSimulatorDevices	189
4.184	GetAllSymbolPropertyNames	189
4.185	GetAllXAxes	190
4.186	GetAllYAxes	190

4.187	GetAnalysisInfo	191
4.188	GetAnalysisLines	191
4.189	GetAnnotationText	192
4.190	GetAxisCurves	192
4.191	GetAxisLimits	193
4.192	GetAxisType	193
4.193	GetAxisUnits	194
4.194	GetChildModulePorts	194
4.195	GetCodecNames	195
4.196	GetColours	195
4.197	GetColourSpec	195
4.198	GetCompatiblePathName	196
4.199	GetComponentValue	196
4.200	GetConfigLoc	197
4.201	GetConnectedPins	197
4.202	GetConvergenceDevNames	198
4.203	GetConvergenceInfo	199
4.204	GetConvergenceNodeNames	200
4.205	GetCurDir	200
4.206	GetCurrentGraph	200
4.207	GetCurrentStepValue	201
4.208	GetCursorCurve	201
4.209	GetCurveAxes	202
4.210	GetCurveAxis	202
4.211	GetCurveCreationIds	203
4.212	GetCurveName	203
4.213	GetCurves	203
4.214	GetCurveVector	204
4.215	GetDatumCurve	204
4.216	GetDeviceDefinition	205
4.217	GetDeviceInfo	206
4.218	GetDeviceParameterNames	207
4.219	GetDevicePins	208
4.220	GetDeviceStats	209
4.221	GetDisabledInstances	209
4.222	GetDivisionLabels	210
4.223	GetDotFilePath	210
4.224	GetDotParamNames	211
4.225	GetDotParamValue	211
4.226	GetDriveType	212
4.227	GetEmbeddedFileName	212
4.228	GetEnvVar	213
4.229	GetEthernetAddresses	213
4.230	GetFillLines	213
4.231	GetFailedNodesFromIterNum	214
4.232	GetFile	215
4.233	GetFileCD	215
4.234	GetFileDir	215
4.235	GetFileExtensions	216
4.236	GetFileInfo	217
4.237	GetFileSave	217
4.238	GetFileVersionStamp	218
4.239	GetFileViewerSelectedDirectories	218
4.240	GetFileViewerSelectedFiles	219
4.241	GetFirstSelectedElementOfType	219
4.242	GetFonts	220

4.243	GetFontSpec	220
4.244	GetFreeDiskSpace	221
4.245	GetGraphFromWindow	221
4.246	GetGraphObjects	222
4.247	GetGraphObjPropNames	222
4.248	GetGraphObjPropValue	223
4.249	GetGraphObjPropValues	224
4.250	GetGraphTabs	224
4.251	GetGraphTitle	225
4.252	GetGridAxes	225
4.253	GetGridCurves	226
4.254	GetGroupAnalysisInfo	226
4.255	GetGroupFromAnalysisId	227
4.256	GetGroupInfo	228
4.257	GetGroupStepParameter	228
4.258	GetGroupStepVals	229
4.259	GetGUID	229
4.260	GetHighlightedWidgetId	229
4.261	GetHostId	230
4.262	GetHttpContentSize	230
4.263	GetInstanceBounds	231
4.264	GetInstanceConvergenceInfo	232
4.265	GetInstanceParamValues	232
4.266	GetInstancePinLocs	233
4.267	GetInstsAtPoint	234
4.268	GetInternalDeviceName	235
4.269	GetKeyDefs	235
4.270	GetKnownFolderPath	236
4.271	GetLaplaceErrorMessage	237
4.272	GetLastCommand	237
4.273	GetLastError	238
4.274	GetLastGraphObjectAdded	238
4.275	GetLegendProperties	239
4.276	GetLibraryModels	240
4.277	GetLicenseInfo	240
4.278	GetLicenseStats	241
4.279	GetLine	241
4.280	GetListSelected	242
4.281	GetListUnselected	242
4.282	GetLongPathName	243
4.283	GetMaxCores	243
4.284	GetMD5	244
4.285	GetMD5String	244
4.286	GetMenuItems	244
4.287	GetModelFiles	245
4.288	GetModelLibraryErrors	245
4.289	GetModelName	246
4.290	GetModelParameterNames	246
4.291	GetModelParameters	247
4.292	GetModelParameterValues	247
4.293	GetModelType	248
4.294	GetModifiedStatus	248
4.295	GetNamedSymbolPins	249
4.296	GetNamedSymbolPropNames	249
4.297	GetNamedSymbolPropValue	250
4.298	GetNearestNet	251

4.299	GetNextDefaultStyleName	251
4.300	GetNodeConvergenceInfo	251
4.301	GetNodeNames	252
4.302	GetNonDefaultOptions	252
4.303	GetNumCurves	253
4.304	GetOpenSchematics	253
4.305	GetOption	254
4.306	GetPath	254
4.307	GetPlatformFeatures	255
4.308	GetPrinterInfo	255
4.309	GetPrintValues	256
4.310	GetReadOnlyStatus	256
4.311	GetRegistryClassesRootKeys	256
4.312	GetSchematicFileVersion	257
4.313	GetSchematicTabs	257
4.314	GetSchematicVersion	258
4.315	GetSchemTitle	259
4.316	GetSelectedAnnotationText	259
4.317	GetSelectedCurves	259
4.318	GetSelectedGraphAnno	259
4.319	GetSelectedGraphAnnotations	260
4.320	GetSelectedGrid	261
4.321	GetSelectedStyleNames	261
4.322	GetSelectedXAxis	261
4.323	GetSelectedYAxis	262
4.324	GetShortPathName	262
4.325	GetSimConfigLoc	262
4.326	GetSimetrixFile	263
4.327	GetSIMPLISExitCode	264
4.328	GetSimulationErrors	264
4.329	GetSimulationInfo	265
4.330	GetSimulationSeeds	265
4.331	GetSimulatorEvents	266
4.332	GetSimulatorMode	267
4.333	GetSimulatorOption	267
4.334	GetSimulatorOptionInfo	268
4.335	GetSimulatorOptions	269
4.336	GetSimulatorStats	269
4.337	GetSimulatorStatus	270
4.338	GetSoaDefinitions	271
4.339	GetSoaMaxMinResults	271
4.340	GetSoaOverloadResults	272
4.341	GetSoaResults	273
4.342	GetSymbolArcInfo	273
4.343	GetSymbolFiles	273
4.344	GetSymbolInfo	274
4.345	GetSymbolOrigin	274
4.346	GetSymbolPropertyInfo	275
4.347	GetSymbolPropertyNames	276
4.348	GetSymbols	276
4.349	GetSystemInfo	277
4.350	GetTempFile	277
4.351	GetTextEditorText	278
4.352	GetThreadTimes	278
4.353	GetTimerInfo	279
4.354	GetTitleBlockInfo	279

4.355	GetToolBarDefinition	280
4.356	GetToolButtons	280
4.357	GetTouchstoneErrors	281
4.358	GetUncPath	282
4.359	GetURLFromLocalPath	282
4.360	GetUsedStyles	282
4.361	GetUserFile	283
4.362	GetVecStepParameter	284
4.363	GetVecStepVals	285
4.364	GetWidgetInfo	285
4.365	GetXAxis	286
4.366	GraphImageCapture	286
4.367	GraphImageCaptureAdvanced	287
4.368	GraphImageParameter	288
4.369	GraphLimits	288
4.370	GroupDelay	289
4.371	Groups	289
4.372	GuiType	290
4.373	Hash	290
4.374	HashAdd	290
4.375	HashCreate	291
4.376	HashDelete	292
4.377	HashKeys	292
4.378	HashRemove	293
4.379	HashSearch	294
4.380	HasLogSpacing	294
4.381	HasProperty	295
4.382	HaveFeature	296
4.383	HaveInternalClipboardData	296
4.384	HierarchyHighlighting	297
4.385	HighlightedNets	297
4.386	Histogram	297
4.387	HistoryDepthDialog	298
4.388	Iff	299
4.389	IffV	299
4.390	IIR	300
4.391	im	301
4.392	imag	301
4.393	InitRandom	302
4.394	InputGraph	302
4.395	InputSchem	303
4.396	Instances	303
4.397	InstNets	304
4.398	InstNets2	304
4.399	InstPins	305
4.400	InstPoints	306
4.401	InstProps	307
4.402	integ	308
4.403	Interp	309
4.404	IsComplex	310
4.405	IsComponent	310
4.406	IsDocumented	310
4.407	IsFileOfType	311
4.408	IsFullPath	311
4.409	IsImageFile	312
4.410	IsModelFile	312

4.411	IsNum	313
4.412	IsOptionMigrateable	313
4.413	IsSameFile	313
4.414	IsScript	314
4.415	IsStr	314
4.416	IsTextEditor	314
4.417	IsTextEditorModified	315
4.418	Join	315
4.419	JoinStringArray	317
4.420	JoinVectors	317
4.421	length	318
4.422	ListDirectory	318
4.423	ListSchemProps	319
4.424	ln	319
4.425	LoadFile	320
4.426	LoadSensitivityReport	320
4.427	LoadTouchstone	321
4.428	Locate	322
4.429	log	322
4.430	log10	322
4.431	LPF	323
4.432	mag	324
4.433	magnitude	324
4.434	makecomplexvec	324
4.435	MakeDir	325
4.436	MakeLogicalPath	325
4.437	MakeString	326
4.438	makevec	326
4.439	ManageDataGroupsDialog	327
4.440	ManageMeasureDialog	328
4.441	MapArray	328
4.442	max	329
4.443	maxidx	329
4.444	Maxima	330
4.445	Maximum	331
4.446	mean	331
4.447	Mean1	332
4.448	MeasureDialog	332
4.449	MessageBox	333
4.450	Mid	334
4.451	min	335
4.452	minidx	335
4.453	Minima	335
4.454	Minimum	336
4.455	MkVec	337
4.456	MLRidgeRegressionFit	337
4.457	MLSplineFit	339
4.458	MLVector	340
4.459	ModelLibsChanged	340
4.460	MSWReadHeader	340
4.461	Navigate	341
4.462	NearestInst	341
4.463	NetName	342
4.464	NetNames	342
4.465	NetWires	343
4.466	NewPassiveDialog	344

4.467	NewValueDialog	345
4.468	norm	346
4.469	NumberSelectedAnnotations	347
4.470	NumDivisions	347
4.471	NumElems	348
4.472	OpenEchoFile	348
4.473	OpenFile	349
4.474	OpenPDFPrinter	349
4.475	OpenPrinter	349
4.476	OpenSchem	350
4.477	OpenSchematic	352
4.478	OptimiserAnalysisLine	354
4.479	OptimiserApplyFailedRun	354
4.480	OptimiserApplySpecification	355
4.481	OptimiserCloseDef	356
4.482	OptimiserCreateFromXML	357
4.483	OptimiserCreateFromXMLString	357
4.484	OptimiserErrorMessage	358
4.485	OptimiserFinish	358
4.486	OptimiserGetDataObject	359
4.487	OptimiserGetDataObjectNames	360
4.488	OptimiserGetInfo	360
4.489	OptimiserGetIteration	361
4.490	OptimiserGetOptions	362
4.491	OptimiserGetParameters	363
4.492	OptimiserGetParameterValues	363
4.493	OptimiserGetSpecification	364
4.494	OptimiserLoadWidgetFromXML	365
4.495	OptimiserParameterLine	365
4.496	OptimiserResults	366
4.497	OptimiserRunning	366
4.498	OptimiserSimulatorGetDef	367
4.499	OptimiserSimulatorGetInfo	367
4.500	OptimiserSimulatorResults	368
4.501	OptimiserSimulatorStatus	368
4.502	OptimiserSimulatorUserMessage	369
4.503	OptimiserSimulatorWriteXMLString	370
4.504	OptimiserStart	370
4.505	OptimiserStatus	371
4.506	OptimiserSuccess	372
4.507	OptimiserUserMessage	373
4.508	OptimiserWidgetCreateOptDef	374
4.509	OptimiserWidgetGetDataObject	374
4.510	OptimiserWidgetGetDataObjectNames	374
4.511	OptimiserWidgetGetInfo	375
4.512	OptimiserWidgetPreFlightChecks	375
4.513	OptimiserWidgetWriteCommandFile	376
4.514	OptimiserWidgetWriteXML	376
4.515	OptimiserWidgetWriteXMLString	377
4.516	OptimiserWriteCommandFile	377
4.517	OptimiserWriteHTMLReport	378
4.518	OptimiserWriteXML	379
4.519	OptimiserWriteXMLString	380
4.520	Parse	381
4.521	ParseAnalysis	381
4.522	ParseEscape	382

4.523	ParseLaplace	383
4.524	ParseParameterString	384
4.525	ParseParameterString2	385
4.526	ParseProbeExpression	387
4.527	ParseSIMPLISInit	388
4.528	ParseSpiceLines	388
4.529	PathEqual	388
4.530	PerCycleTiming	389
4.531	PerCycleValue	391
4.532	ph	393
4.533	phase	394
4.534	phase_rad	394
4.535	PhysType	395
4.536	PinName	396
4.537	PrepareSetComponentValue	396
4.538	PreProcessNetlist	398
4.539	Probe	399
4.540	ProcessingAccelerator	399
4.541	ProcessingDragAndDrop	400
4.542	ProcessingGuiAction	400
4.543	Progress	401
4.544	PropFlags	401
4.545	PropFlags2	402
4.546	PropFlagsAll	403
4.547	PropFlagsAnnotations	405
4.548	PropFlagsWires	406
4.549	PropOverrideStyle	407
4.550	PropValue	407
4.551	PropValues	408
4.552	PropValues2	409
4.553	PropValuesAll	410
4.554	PropValuesAnnotations	411
4.555	PropValuesWires	412
4.556	PutEnvVar	413
4.557	PWLCurveFit	413
4.558	PWLDialog	414
4.559	QueryData	416
4.560	RadioSelect	418
4.561	Range	419
4.562	re	419
4.563	ReadClipboard	419
4.564	ReadConfigCollection	420
4.565	ReadConfigSetting	420
4.566	ReadF11Analyses	421
4.567	ReadF11Options	422
4.568	ReadFile	423
4.569	ReadIniKey	423
4.570	ReadRegSetting	424
4.571	ReadSchemProp	425
4.572	ReadSIMPLISF11Data	426
4.573	ReadSpiceFile	426
4.574	ReadTextEditorProp	427
4.575	ReadTouchstone	427
4.576	real	428
4.577	Ref	428
4.578	RefName	429

4.579	RegExContains	429
4.580	RegExIsValid	431
4.581	RegExMatch	432
4.582	RegExp	434
4.583	RegExReplace	434
4.584	RegExSearch	436
4.585	RegExSearchAll	438
4.586	RegExSplit	440
4.587	RegExStrStr	441
4.588	RelativePath	443
4.589	RemapDevice	443
4.590	RemoveConfigCollection	445
4.591	RemoveModelFile	445
4.592	RemoveSymbolFiles	445
4.593	ResolveGraphTemplate	446
4.594	ResolveTemplate	447
4.595	RestartTranDialog	448
4.596	Rms	448
4.597	RMS1	449
4.598	rnd	449
4.599	RootSumOfSquares	449
4.600	rt	450
4.601	SaveSpecialDialog	450
4.602	Scan	451
4.603	ScanEscape	451
4.604	ScriptName	453
4.605	Search	453
4.606	SearchModels	454
4.607	Seconds	455
4.608	Select2Dialog	455
4.609	SelectAnalysis	456
4.610	SelectColourDialog	456
4.611	SelectColumns	457
4.612	SelectCount	457
4.613	SelectDevice	458
4.614	SelectDialog	459
4.615	SelectedProperties	460
4.616	SelectedStyleInfo	460
4.617	SelectedWires	461
4.618	SelectFontDialog	461
4.619	SelectRows	461
4.620	SelectSIMPLISAnalysis	462
4.621	SelectSymbolDialog	463
4.622	SelGraph	464
4.623	SelSchem	464
4.624	SetComponentValue	465
4.625	SetDifference	466
4.626	SetInstanceParamValue	467
4.627	SetIntersect	468
4.628	SetModelParamValue	469
4.629	SetPropertyStyles	470
4.630	SetReadOnlyStatus	470
4.631	SetSymmetricDifference	471
4.632	SetUnion	472
4.633	Shell	472
4.634	ShellExecute	474

4.635	ShiftRef	475
4.636	sign	476
4.637	SimetrixFileInfo	476
4.638	SIMPLISRunStatus	477
4.639	SIMPLISSearchIdx	477
4.640	SimulationHasErrors	478
4.641	sin	478
4.642	sin_deg	479
4.643	sinh	479
4.644	Sleep	479
4.645	Sort	480
4.646	SortIdx	480
4.647	SortIdx2	481
4.648	SourceDialog	482
4.649	SpectrumUniv	482
4.650	SplitPath	483
4.651	SplitString	484
4.652	SprintfNumber	484
4.653	sqrt	485
4.654	Str	485
4.655	StringLength	486
4.656	StringStartsWith	486
4.657	StrStr	486
4.658	StyleInfo	487
4.659	StyleLineTypes	488
4.660	StyleNames	488
4.661	SubstChar	489
4.662	SubstProbeExpression	489
4.663	SubstString	490
4.664	sum	490
4.665	SumNoise	491
4.666	SupportedReadFormats	491
4.667	SupportedWriteFormats	492
4.668	SxUUID	492
4.669	SymbolInfoDialog	492
4.670	SymbolLibraryManagerDialog	493
4.671	SymbolName	494
4.672	SymbolNames	494
4.673	SymbolPinOrder	495
4.674	SymbolPinPoints	496
4.675	SystemValue	496
4.676	SystemValuePath	496
4.677	SystemWidgetExistsInSelectedWindow	497
4.678	TableDialog	497
4.679	TableEditor	498
4.680	tan	499
4.681	tan_deg	500
4.682	tanh	500
4.683	TemplateGetPropValue	500
4.684	TemplateResolve	501
4.685	TextEditorHasComments	501
4.686	ThdWeight	501
4.687	TickCount	502
4.688	Time	502
4.689	ToLower	503
4.690	ToUpper	503

4.691	TranslateLogicalPath	503
4.692	TreeListDialog	504
4.693	True	505
4.694	Truncate	505
4.695	TwoFileSelectionDialog	506
4.696	UD	511
4.697	UngroupCurve	511
4.698	Unif	512
4.699	Units	512
4.700	unitvec	513
4.701	UpDownDialog	514
4.702	UserParametersDialog	515
4.703	Val	515
4.704	ValueDialog	516
4.705	Vec	517
4.706	vector	518
4.707	VectorsInGroup	518
4.708	VersionInfo	519
4.709	ViewFormattedFile	519
4.710	ViewFormattedText	520
4.711	WAV_CloseFile	520
4.712	WAV_OpenFile	521
4.713	WAV_ReadData	521
4.714	WAV_WriteFile	522
4.715	WC	523
4.716	WC2	524
4.717	WirePoints	524
4.718	Wires	525
4.719	WM_CanRevertToSaved	526
4.720	WM_GetAllVisibleContentWidgetNames	526
4.721	WM_GetCentralWidgetGeometry	527
4.722	WM_GetContentWidgetNames	528
4.723	WM_GetContentWidgetSessionInfo	528
4.724	WM_GetContentWidgetsLayout	528
4.725	WM_GetContentWidgetsOfType	529
4.726	WM_GetContentWidgetTypes	530
4.727	WM_GetCurrentWidgetInfo	530
4.728	WM_GetCurrentWindowName	532
4.729	WM_GetLastAccessedContentWidget	532
4.730	WM_GetNumberModifiedEditors	532
4.731	WM_GetPrimaryWindowName	532
4.732	WM_GetSystemWidgetSessionInfo	533
4.733	WM_GetSystemWidgetsLayout	533
4.734	WM_GetWidgetInstanceInfo	534
4.735	WM_GetWindowGeometry	534
4.736	WM_GetWindowNames	535
4.737	WM_NumberContentWidgets	535
4.738	WM_NumberSystemWidgets	535
4.739	WriteConfigSetting	536
4.740	WriteF11Lines	536
4.741	WriteF11Options	537
4.742	WriteIniKey	538
4.743	WriteRawData	539
4.744	WriteRegSetting	539
4.745	WriteSchemProp	540
4.746	XCursor	541

4.747	XDatum	541
4.748	XFromY	542
4.749	XMLCountElements	542
4.750	XMLGetAttribute	542
4.751	XMLGetElements	543
4.752	XMLGetText	543
4.753	XMLToString	544
4.754	XY	544
4.755	XYOrdered	544
4.756	YCursor	545
4.757	YDatum	545
4.758	YFromX	545
5	Command Summary	547
5.1	Commands by Application	557
5.1.1	File	557
5.1.2	Graph	558
5.1.3	Lib	558
5.1.4	Miscellaneous	558
5.1.5	Optimiser	558
5.1.6	Printing	558
5.1.7	Schematic	558
5.1.8	Simulator	560
5.1.9	Text Editor	560
5.1.10	User Interface	560
5.1.11	Vectors/Groups	561
5.1.12	XML Processing	561
6	Command Reference	562
6.1	Abort	562
6.2	AbortSIMPLIS	562
6.3	About	562
6.4	AddAnnotationText	562
6.5	AddArc	563
6.6	AddCirc	563
6.7	AddCurveMarker	564
6.8	AddDoubleClickAction	564
6.9	AddFileViewMenuItem	565
6.10	AddFloodFill	565
6.11	AddFreeText	566
6.12	AddGlobalStyle	567
6.13	AddImage	567
6.14	AddImageScript	568
6.15	AddLegend	568
6.16	AddLegendProp	569
6.17	AddPin	570
6.18	AddProp	571
6.19	AddProperty	573
6.20	AddSeg	573
6.21	AddSymbolProperty	574
6.22	AddTextBox	574
6.23	AddTitleBlock	575
6.24	AlignText	575
6.25	Anno	576
6.26	AppendGroup	576
6.27	AppendTextWindow	577

6.28	Arguments	577
6.29	BuildDefaultOptions	578
6.30	Cancel	578
6.31	CaptureWaveformImage	578
6.32	Cd	578
6.33	ChangeArcAttributes	578
6.34	ChangeSelectedStyleNames	579
6.35	ChangeStyle	579
6.36	ChangeSymbolProperty	579
6.37	ClearMessageWindow	580
6.38	Close	580
6.39	CloseGraphSheet	580
6.40	CloseLinkedRun	580
6.41	ClosePrinter	581
6.42	CloseSchem	581
6.43	CloseSheet	581
6.44	CloseSimplisStatusBox	582
6.45	CloseTextEditor	582
6.46	CollectGarbage	582
6.47	CombineMenu	582
6.48	CompareSymbolLibs	583
6.49	CompressXML	583
6.50	ConvertBinaryGraph	584
6.51	Copy	584
6.52	CopyClipGraph	584
6.53	CopyClipSchem	585
6.54	CopyFile	586
6.55	CopyLocalSymbol	586
6.56	CreateFont	586
6.57	CreateGroup	587
6.58	CreateRunningDialog	587
6.59	CreateSym	588
6.60	CreateToolBar	588
6.61	CreateToolButton	590
6.62	CursorMode	591
6.63	Curve	592
6.64	CurveEditCopy	594
6.65	DecompressXML	594
6.66	DefButton	595
6.67	DefineToolBar	596
6.68	DefKey	596
6.69	DefMenu	599
6.70	Del	602
6.71	DelCrv	602
6.72	Delete	603
6.73	DeleteAxis	603
6.74	DeleteGlobalStyle	603
6.75	DeleteGraphAnno	603
6.76	DeleteGraphObject	604
6.77	DeleteShortWires	604
6.78	DeleteSymbolProperty	604
6.79	DeleteToolBar	605
6.80	DelGroup	605
6.81	DelLegendProp	606
6.82	DelMenu	606
6.83	DelProp	607

6.84	DelSym	607
6.85	DestroyRunningDialog	608
6.86	Detach	608
6.87	Discard	608
6.88	Display	609
6.89	DrawArc	609
6.90	DrawArrow	609
6.91	DrawPin	610
6.92	Echo	610
6.93	EditColour	611
6.94	EditCopy	611
6.95	EditCut	612
6.96	EditFile	612
6.97	EditFont	612
6.98	EditGroupTitle	612
6.99	EditPaste	613
6.100	EditPin	613
6.101	EndAllInteractions	613
6.102	EndSym	613
6.103	Execute	613
6.104	ExecuteMenu	614
6.105	FileViewCleanUpFileWatchers	615
6.106	FloodFillSymbol	615
6.107	Focus	615
6.108	FocusCommandShell	615
6.109	FocusShell	616
6.110	GroupSelected	616
6.111	Help	616
6.112	HideCurve	616
6.113	HighlightCurve	617
6.114	HighlightWidget	617
6.115	Hint	617
6.116	HourGlass	618
6.117	ImportSymbol	618
6.118	Inst	619
6.119	KeepGroup	620
6.120	Let	621
6.121	Listing	621
6.122	ListModels	621
6.123	ListOptions	622
6.124	LoadModelIndex	622
6.125	LoadSimulatorStyleSheet	622
6.126	LoadStyleSheet	623
6.127	LocalDeclAll	623
6.128	MakeAlias	624
6.129	MakeCatalog	624
6.130	MakeSymbolScript	624
6.131	MakeTree	625
6.132	MCD	625
6.133	MD	625
6.134	Message	625
6.135	MessageBox	626
6.136	Move	626
6.137	MoveCurve	626
6.138	MoveFile	627
6.139	MoveMenu	627

6.140	MoveProperty	627
6.141	Netlist	628
6.142	NewAnnotation	629
6.143	NewAxis	630
6.144	NewBasicTextEditor	630
6.145	NewFileView	630
6.146	NewGraphWindow	631
6.147	NewGrid	631
6.148	NewLabel	631
6.149	NewLogicDefinitionEditor	632
6.150	NewNetlist	632
6.151	NewPartSelector	632
6.152	NewPlot	632
6.153	NewPrinterPage	633
6.154	NewSchem	633
6.155	NewScript	633
6.156	NewStyle	633
6.157	NewSymbol	634
6.158	NewVerilogA	634
6.159	NewVerilogHDL	635
6.160	NoPaint	635
6.161	NoUndo	635
6.162	OpenAsciiFile	636
6.163	OpenBasicTextEditor	636
6.164	OpenDirectory	637
6.165	OpenExternalFile	637
6.166	OpenGraph	637
6.167	OpenGroup	637
6.168	OpenLinkedRun	639
6.169	OpenLogicDefinitionEditor	639
6.170	OpenNetlist	640
6.171	OpenPrinter	640
6.172	OpenRawFile	641
6.173	OpenSchem	641
6.174	OpenScript	642
6.175	OpenSimplisStatusBox	643
6.176	OpenVerilogA	643
6.177	OpenVerilogHDL	643
6.178	OpenWebPage	644
6.179	OptimiserCreate	644
6.180	OptimiserRequestFinish	644
6.181	OptionsDialog	645
6.182	Pan	645
6.183	PasteGraphImageToSchematic	645
6.184	Pause	645
6.185	PlaceCursor	645
6.186	Plot	646
6.187	PreProcessNetlist	646
6.188	PrintGraph	646
6.189	PrintSchematic	647
6.190	Probe	648
6.191	Prop	648
6.192	Protect	651
6.193	Quit	651
6.194	RD	651
6.195	ReadLogicCompatibility	651

6.196	RebuildSymbols	653
6.197	Redirect	653
6.198	RedirectMessages	653
6.199	RedrawGraph	654
6.200	RegisterUserFunction	654
6.201	RenameLibs	655
6.202	RenameMenu	655
6.203	RepeatLastMenu	655
6.204	ReplayTraces	656
6.205	Reset	656
6.206	ResizeWindow	656
6.207	RestartTran	656
6.208	RestoreCommandShell	657
6.209	RestoreDefaultStyles	657
6.210	Resume	657
6.211	RotInst	657
6.212	Run	658
6.213	RunAsync	660
6.214	RunCurrentScript	661
6.215	Save	661
6.216	SaveAs	661
6.217	SaveGraph	662
6.218	SaveGroup	662
6.219	SaveRhs	663
6.220	SaveSnapShot	663
6.221	SaveSymbol	664
6.222	SaveSymlib	664
6.223	SaveTextEditor	665
6.224	SaveTextEditorAs	665
6.225	SchematicEnableFileWatcher	665
6.226	SchematicFileWatcherIgnoreChanges	666
6.227	SchematicFileWatcherWatchChanges	666
6.228	ScreenShotWindow	666
6.229	ScriptAbort	666
6.230	ScriptPause	666
6.231	ScriptResume	667
6.232	ScriptStep	667
6.233	Select	667
6.234	SelectCurve	668
6.235	SelectGraph	668
6.236	SelectLegends	669
6.237	SelectSchematic	669
6.238	SelectSimulator	669
6.239	SelectSymbolPin	669
6.240	SelectWireConnected	670
6.241	Set	670
6.242	SetAnnotationTextPosition	670
6.243	SetCurveName	671
6.244	SetDefaultEncoding	671
6.245	SetDisable	672
6.246	SetGraphAnnoProperty	672
6.247	SetGroup	673
6.248	SetHighlight	673
6.249	SetOrigin	674
6.250	SetPinPrefix	674
6.251	SetPinSuffix	674

6.252	SetReadOnly	675
6.253	SetRef	675
6.254	SetSnapGrid	675
6.255	SetStyleColour	675
6.256	SetSymbolFillStyle	676
6.257	SetSymbolOriginVisibility	676
6.258	SetUnits	676
6.259	Shell	677
6.260	ShellOld	678
6.261	Show	678
6.262	ShowCurve	679
6.263	ShowSimulatorWindow	679
6.264	SizeGraph	680
6.265	SwitchModelSection	680
6.266	TemplateEditProperty	680
6.267	TemplateSetValue	681
6.268	TextEditorCommentLines	681
6.269	TextEditorFileWatcherIgnoreChanges	681
6.270	TextEditorFileWatcherWatchChanges	681
6.271	TextEditorFind	682
6.272	TextEditorFindNext	682
6.273	TextEditorGoToLine	682
6.274	TextEditorUncommentLines	682
6.275	TextWin	682
6.276	Title	682
6.277	TitleSchem	683
6.278	Trace	683
6.279	UndoGraphZoom	683
6.280	UngroupSelected	684
6.281	UnHighlightCurves	684
6.282	UnLet	684
6.283	Unprotect	684
6.284	Unselect	685
6.285	UnSet	685
6.286	Unzip	685
6.287	UpdateAllSymbols	686
6.288	UpdateAnnotationText	686
6.289	UpdateDefaultStyle	686
6.290	UpdateGlobalStyle	687
6.291	UpdateGraphColours	687
6.292	UpdateProperties	688
6.293	UpdateRunningDialog	688
6.294	UpdateStyleInfo	688
6.295	UpdateSymbol	689
6.296	UpdateSystemStyleInfo	689
6.297	UpdateTitleBlock	689
6.298	UseGlobalStyles	690
6.299	ViewFile	690
6.300	WebOpen	690
6.301	Wire	690
6.302	WireMode	691
6.303	WM_CloseAllSystemWidgets	691
6.304	WM_CloseNonPrimaryWindows	691
6.305	WM_CloseSheet	691
6.306	WM_ProgressWindowClose	692
6.307	WM_ProgressWindowCloseAll	692

6.308	WM_ProgressWindowCreate	692
6.309	WM_ProgressWindowReport	693
6.310	WM_RevertToSaved	693
6.311	WM_Undock	693
6.312	WriteImportedModels	694
6.313	XMLAddAttribute	694
6.314	XMLAddElement	694
6.315	XMLClose	694
6.316	XMLGoUpLevel	695
6.317	XMLNew	695
6.318	XMLOpenElement	695
6.319	XMLOpenFile	696
6.320	XMLSaveFile	696
6.321	Zip	696
6.322	Zoom	696
7	Applications	698
7.1	User Interface	698
7.1.1	User Defined Key and Menu Definitions	698
7.1.2	Rearranging or Renaming the Standard Menus	698
7.1.3	Menu Shortcuts	699
7.1.4	Editing Schematic Component Values	699
7.1.5	Modifying Internal Scripts	699
7.2	Custom Curve Analysis	699
7.2.1	Adding New Functions	699
7.2.2	'measure', 'measure_span' Scripts	700
7.2.3	An Example: The 'Mean' Function	700
7.3	Automating Simulations	700
7.3.1	Overview	700
7.3.2	Running the Simulator	701
7.3.3	Changing Component Values or Test Conditions	701
7.3.4	Organising Data Output from Automated Runs	702
7.3.5	An Advanced Example - Reading Values from a File	702
7.4	Optimiser	705
7.4.1	Accessing Optimiser Definitions	706
7.4.2	Running an Optimiser Session	707
7.4.3	Postprocessing Optimiser Results	707
7.5	Schematic Symbol Script Definition	708
7.5.1	Defining New Symbol	708
7.5.2	Symbol Definition Format	708
7.5.3	How Symbols are Stored	711
7.6	Data Import and Export	711
7.6.1	Importing Data	711
7.6.2	Exporting Data	711
7.6.3	Launching Other Applications	712
7.6.4	Data Files Text Format	712
7.7	Graph Objects	713
7.7.1	Overview	713
7.7.2	Object Types	713
7.7.3	Properties	714
7.7.4	Graph Object Identifiers - the "ID"	714
7.7.5	Symbolic Values	715
7.7.6	Objects and Their Properties	715
7.7.7	Graph Co-ordinate Systems	730
7.8	Event Scripts	730
7.8.1	on_graph_anno_doubleclick	730

7.8.2	on_accept_file_drop	730
7.8.3	on_schem_double_click	731
7.9	User Defined Script Based Functions	731
7.9.1	Overview	731
7.9.2	Defining the Function	731
7.9.3	Registering the Script	731
7.9.4	Example	731
7.10	User Defined Binary Functions	732
7.10.1	Overview	732
7.10.2	Documentation	732
7.11	Non-interactive and Customised Printing	732
7.11.1	Overview	732
7.11.2	Procedure	733
7.11.3	Example	733
7.12	Schematic Template Scripts	734
7.12.1	Overview	734
7.12.2	Defining a Symbol for a Template Script	734
7.12.3	When is the Template Script Called?	734
7.12.4	The Template Script	734
7.12.5	Template Commands and Functions	735
7.13	Creating and Modifying Toolbars	735
7.13.1	Modifying Existing Toolbars and Buttons	735
7.13.2	Redefining Button Commands	736
7.13.3	Defining New Buttons and Editing Buttons	736
7.13.4	Creating New Toolbars	737
7.13.5	Pre-defined Buttons	737
7.14	Custom Dialog Boxes	741
7.14.1	Overview	741
7.14.2	Starting “Designer”	741
7.14.3	Developing Dialogs	741
7.14.4	The Widgets	742
7.14.5	Using Geometry Management	745
7.14.6	Examples	745
7.14.7	ExecuteDialog Function	746
7.14.8	Performance	747
7.15	Pre-defined Buttons	747

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 600 functions and 300 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

2.1 A Tutorial

2.1.1 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 | New | Script** this will open the in built script editor. Type:

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

Now save the text to a file called *hello.sxscr*. To execute the script, click on the Run toolbar button or 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. 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](#).

2.1.2 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 | New Script**. This will open a text editor with the current directory set to the SCRIPT. Type in the following:

```

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.sxscr*. 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 two lines carry out the simulation and in fact something similar is done each time a simulation is run using the menu or F9 key. `Netlist design.net` generates a netlist of the circuit and saves it in a file called `design.net`. Then `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* `umsum`. `sum` is evaluated and the result replaces expression and the braces. See [Braced Substitutions](#) for more information.

2.1.3 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()` and `Probe()` and the command `Probe`.

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, `etname()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 the 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()`, `NearestInst()`, `PinName()` and `Branch()`.

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.

2.1.4 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.

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()`. 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.

2.2 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.

2.2.1 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
 foreach
 foreachdiv
 if
 loop
 next
 script
 step
 then
 to
 while

2.2.2 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} .

2.2.3 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.

Example 1: Real

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

Example 2: Complex

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

Example 3: String

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

2.2.4 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.

2.2.5 Special Characters

Some characters have a special meaning and if entered into a string literal will not work correctly. Characters affected are newline, tab, semi-colon, single and double quotation and open and close brace characters.

Open and close brace characters (`{` and `}`) and semi-colon (`;`) may be included in a string literal by enclosing the whole string with *double* quotation marks. (There is more information here [Quotes: Single or Double](#)).

Single and double quotation marks can be included by doubling them up. However, this can be inconvenient and an alternative method is to assign a variable with the special character using the `Chr()` function. This method is also the only way to enter a tab character into a literal string. For example:

```
Let tab = chr(9)
Let string = 'This is a tab ' & tab & ' character'
```

This method can be used to enter new line characters (`chr(10)`) and also single quotes (`chr(39)`), double quotes (`chr(34)`) and semi-colons (`chr(59)`)

2.2.6 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 in the section on [Bracketed Lists](#). 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.

2.2.7 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.

2.2.8 Empty Values

Many functions return *empty* values (also known as empty vectors or zero-length 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. Most other functions and operators will yield an error if presented with an empty value. The exceptions are

```
JoinStringArray()
JoinVectors()
NumElems()
SetDifference()
SetIntersect()
SetSymmetricDifference()
SetUnion()
```

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

2.2.9 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.

2.2.10 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 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.

2.3 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*) and *if statements*, *while statements* and *for statements*. This section describes expression syntax and how they are evaluated.

2.3.1 Operators

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 - + 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.

2.3.2 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:

```
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](#).

2.3.3 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
```

2.3.4 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.

2.3.5 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.

2.3.6 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.

2.4 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 command names listed in the [Command Reference](#).

There are six types of statement. These are:

- command statement
- if statement
- while statement
- for statement
- foreach statement
- foreachdiv statement
- exit statement
- script statement

2.4.1 Commands

Commands begin with one of the names of commands listed in the [Command Summary](#). 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.

2.4.2 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.

2.4.3 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**.

2.4.4 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`.

2.4.5 For Statement

These are of the form:

```
for variable=expression1 to expression2 [ step expression3]
  statement
next [variable]
```

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.

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
```

2.4.6 Foreach Statement

These are of the form:

```
foreach variable array_expression
  statement
next
```

The foreach statement executes `statement` for every item in the array `array_expression`. The item in the array is assigned to the variable `variable`

Notes

`array_expression` is evaluated before the loop is executed and assigned to a hidden variable. So changes to `array_expression` will not affect the execution of the loop.

If `array_expression` is actually a scalar, the loop will execute once with the scalar value assigned to variable.

If `array_expression` is a multi-division vector with 2 or more divisions, the loop will execute for each division and the division will be assigned to `variable`.

A [foreachdiv statement](#) can alternatively be used. This will execute only once for a single division vector.

2.4.7 Foreachdiv Statement

These are of the form:

```
foreachdiv variable array_expression
    statement
next
```

The foreachdiv statement works with [Multi-division vectors](#).

The foreach statement executes `statement` for every division in the array `array_expression`. The division in the array is assigned to the variable `variable`.

2.4.8 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.

2.4.9 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.

2.5 Accessing Simulation Data

2.5.1 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 [aliases](#).

Each of these is described in the following sections.

2.5.2 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](#)). 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 **Simulator | Manage Data Groups**. Further the number of groups kept can be changed with the GroupPersistence option. See *User's Manual/Sundry Topics/Options/UsingtheSetandUnsetcommands/List of Options* 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 `tran1` 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 **Simulator | Manage Data Groups** menu or with the [SetGroup](#). 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 in the [aliases section](#).

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 function `Vec()` can also be used for this purpose.

2.5.3 Multi-division Vectors

Multi-step runs such as Monte Carlo produce multiple vectors representing the same physical quantity. In SIMetrix these multiple vectors are stored as a single entity in what is known as 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()`.

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

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>avg()</code>	Multi			
<code>cos()</code>	Multi			
<code>cosh()</code>	Multi			
<code>CyclePeriod()</code>	Multi	Multi		

Function name	Arg 1	Arg 2	Arg 3	Arg 4
cos_deg	Multi			
db()	Multi			
DefineFourierDialog()	X	Scalar		
diff()	Multi			
Execute()	X	Multi	Multi	Multi
erf()	Multi			
exp()	Multi			
fft()	Multi	X		
FIR()	Multi	X	X	
Fourier()	Multi	X	X	X
FourierOptionsDialog()	X	Scalar		
FourierWindow()	Multi	X	X	
gamma()	Multi			
GetVecStepParameter()	Scalar			
GetVecStepVals()	Scalar			
GroupDelay()	Multi			
HasLogSpacing()	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			
ln()	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	
Minimum()	Multi	X	X	

Function name	Arg 1	Arg 2	Arg 3	Arg 4
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			
sinh()	Multi			
sin_deg	Multi			
sqrt()	Multi			
SumNoise()	Multi	X	X	
tan()	Multi			
tanh()	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](#). Also the [XY\(\)](#) function may be used to compose a vector containing a reference.

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](#).

2.6 User Interface to Scripts

2.6.1 Dialog Boxes

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

Function name	Comment
AddRemoveDialog()	Add or remove items to or from a list
BoolSelect()	Up to 6 check boxes
ChooseDir()	Select a directory
EditObjectPropertiesDialog()	Read/Edit a list of property names and values
EditSelect()	Up to 6 edit boxes
EnterTextDialog()	Enter multi line text
GetSimetrixFile()	Get file name of pre-defined type
GetUserFile()	Get file name (general purpose)
InputGraph()	Input text for graph
InputSchem()	Input text for schematic
NewValueDialog()	General purpose dialog box
RadioSelect()	Up to 6 radio buttons
SelectDialog()	Select item(s) from a list
TableDialog()	Present items in a table
TableEditor()	Present lists of items in a table
TreeListDialog()	Select item from tree structured list
UpDownDialog()	Re order items
UserParametersDialog()	Read/Edit a list of parameter names and values
ValueDialog()	Up to 10 edit boxes for entering values

The above are the general purpose user interface functions. In particular, the function [NewValueDialog\(\)](#) 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](#).

2.6.2 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](#) 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
```

2.7 Errors

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

2.7.1 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.

2.7.2 Execution Errors

These occur when the script executes and are mostly the result of a command execution failure or an expression evaluation failure.

2.8 Executing Scripts

Scripts are executed by typing their file name at the command line, running them from the script editor, or dragging and dropping the file to the Command shell. Additionally, scripts can be assigned to a key or menu. See [User Defined Key and Menu Definitions](#).

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`.
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](#).
3. Search the SCRIPT directory. This is defined by the ScriptDir option setting (see [Set](#)) which can also be accessed in the File Locations tab of the options dialog box. (see **File | Options | General...**).
4. Search the User Script list of directories. This is defined by the UserScriptDir option variable (see [Set](#)). 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](#).

2.8.1 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. 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` 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.

2.8.2 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.

2.8.3 Debugging Scripts

Displaying 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
```

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.

Abort 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.

Resume a Paused Script

Press ctrl-F2

2.8.4 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 | Options | Edit Startup Script** menu item.

2.9 Unsupported Functions and Commands

A very small number of functions and commands are designated as *unsupported*. These are usually functions or commands 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 or command and it is useful to you, please tell technical support - by Email preferably. If a number of users find the function or command useful, we will raise its status to supported.

Chapter 3

Function Summary

The following table lists all functions available.

Function Name	Description
abs	Absolute value
acos	Inverse cosine (radians)
acosh	Inverse hyperbolic cosine (radians)
ACSourceDialog	Displays dialog box intended for the user definition of an AC source
ACSourceDialogStr	Displays dialog box intended for the user definition of an AC source
AddConfigCollection	Adds a list of entries to a named section in the configuration file
AddGraphCrossHair	Adds a new cursor to the current graph
AddModelFiles	Installs Model Files
AddPropertyDialog	User interface function. Open add property dialog for symbol editor
AddRemoveDialog	User interface function. Allows selection of a list of items.
AddRemoveDialogNew	User interface function. Allows selection of a list of items.
AddSymbolFiles	Adds file or files to list of installed symbol library files
AppendSensitivityData	Appends new data to an existing sensitivity XML file
area	Calculates the area under a curve
arg	Phase of argument in degrees. Result always between -180 to 180
arg_rad	Phase of argument in radians. Result always between $-\pi$ to π
Ascii	Returns ASCII code for character

Function Name	Description
asin	Inverse sine (radians)
asinh	Inverse hyperbolic sine (radians)
AssociateModel	Special purpose function for managing parts browser.
atan	Arc Tangent (radians)
atan2	Returns the principal value of the arc tangent of y/x taking into account the sign of both arguments in order to determine the quadrant.
atan2_deg	Returns the principal value of the arc tangent of y/x taking into account the sign of both arguments in order to determine the quadrant. Returns the result in degrees
atan_deg	Arc Tangent (degrees)
atanh	Inverse hyperbolic tangent (radians)
avg	Returns the average of argument
BoolSelect	User interface function. Returns state of up to 6 check boxes
Branch	Returns branch current formula of schematic net nearest cursor
BuildMclogHTML	Builds an HTML Monte Carlo log file from XML input
BuildParameterString	Constructs a string of name=value pairs
BuildSensitivityHTML	Builds a Sensitivity report file from XML input
BuildWorstCaseHTML	Builds a Worst-case report file from XML input
CanOpenFile	Returns TRUE if specified file exists and can be opened for read
ChangeDir	Change current working directory
Char	Returns character from string
CheckLaplaceExpression	Checks a Laplace expression for correctness
ChooseDir	User interface function. Returns user selected path-name.
ChooseDirectory	User interface function. Returns user selected path-name.
Chr	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	Closes a file opened using OpenFile
CloseSchematic	Close a schematic handle opened using OpenSchematic()

Function Name	Description
CloseSchematicTab	Close a schematic using ID
Coll	Collates a multi-division vector
CollateVectors	Returns vector data in an interleaved manner
CommandStatus	Obtain information about the current script execution context
CompareSymbols	Compare two schematic symbols
ComposeDigital	Builds a new vector from a binary weighted combination of digital vectors
ConvertFromBase64	Converts a Base64 expression to string
ConvertHTMLcolourToRGB	Converts a colour definition from HTML format to decimal RGB format
ConvertIsoTimeToUnix	Converts time in form DD/MM/YYYY to Unix Epoch time
ConvertLocalToUnix	Convert file name to UNIX format using '/'
ConvertNumberFromBase64	Converts a Base64 number to string
ConvertNumberToBase64	Converts a number to Base64
ConvertRGBcolourToHTML	Converts a colour definition from Decimal RGB format to HTML format
ConvertToBase64	Converts a string to Base64
ConvertUnixTimeToIso	Converts UNIX time to form YYYY-MM-DD
ConvertUnixToLocal	Convert file name to the local format.
CopyDivisionData	Copies division data from a vector to another vector
CopyTree	Copy a directory tree
CopyURL	Copy a file to or from a location defined by a URL. Supports http, ftp and local files.
cos	Cosine (radians)
cos_deg	Cosine (degrees)
cosh	Hyperbolic cosine (radians)
CountChars	Counts characters in a string
CreateDiodeDialog	Opens a specialised dialog used by the diode model in-circuit parameter extractor
CreateGraphMeasurement	Creates a graph measurement object
CreateLockFile	Create or remove a lock file for specified file.
CreateNewTitleBlockDialog	Displays the title block creation dialog
CreateSharedAxisConnector	Creates a SharedAxis object to connect x-axes
CreateShortcut	Create a shortcut to the specified path

Function Name	Description
CreateTimer	Create a timer to schedule events in the future
CurveEditDialog	Open dialog box to edit curve labels and colours
CurveFit	Compares two vectors and returns a normalised difference value
CurveFitLog	Compares two vectors and returns a normalised difference value. Applies a log function to the x and y data before calculating the difference value
CurveFitLogX	Compares two vectors and returns a normalised difference value. Applies a log function to the x-data before calculating the difference value
CurveFitLogY	Compares two vectors and returns a normalised difference value. Applies a log function to the y-data before calculating the difference value
cv	Alias to GetCurveVector()
CyclePeriod	Returns the time between zero crossing pairs with the same slope direction. It can be used for plotting frequency vs time
Date	Return current system date in string form
db	$dB(x) = 20 \times \log_{10}(\text{mag}(x))$
DCSourceDialog	Opens 'Edit DC Source' dialog box
DCSourceDialogStr	Opens 'Edit DC Source' dialog box
DefineADCDialog	UI function to define generic ADC
DefineArbSourceDialog	UI function to define arbitrary source
DefineBusPlotDialog	Opens a dialog box to allow the user to plot a bus
DefineCounterDialog	UI function to define generic counter
DefineCurveDialog	Opens define curve dialog box
DefineDACDialog	UI function to define generic DAC
DefineFourierDialog	UI function, opens define fourier dialog
DefineFourierProbeDialog	Opens dialog to edit fixed Fourier Probe
DefineIdealTxDialog	UI function to define ideal transformer
DefineLaplaceDialog	UI function to define S-domain transfer function
DefineLogicGateDialog	UI function to define generic logic gate
DefinePerfAnalysisDialog	UI function for defining a performance analysis
DefineRegisterDialog	UI function to define Bus register
DefineRipperDialog	UI function to define schematic bus ripper
DefineSaturableTxDialog	Open dialog box to define a saturable transformer
DefineShiftRegDialog	UI function to define generic shift register

Function Name	Description
DefineSimplisMultiStepDialog	Open dialog box to define SIMPLIS multi-step dialog.
DeleteConfigCollection	Deletes a list of entries in the config file
DeleteTimer	Deletes a timer
DeleteTouchstone	Deletes data loaded by LoadTouchstone()
DeleteTree	Delete an entire directory tree
DeleteTreeProgress	Delete an entire directory tree, while showing a progress box
DelSchemProp	Deletes a schematic window property
DescendDirectories	Returns all directories under the specified directory, recursing through all sub-directories
DescendHierarchy	Analyse schematic hierarchy
DialogDesigner	Simple dialog designer
diff	Return derivative of argument
DirectoryIsWriteable	Tests whether or not a directory can be written to
Distribution	Returns random number with a custom distribution
EditArcDialog	UI function to edit symbol editor arc
EditAxisDialog	UI function, opens edit axis dialog
EditBodePlotProbeDialog	UI function for editing Bode plot fixed probes
EditBodePlotProbeDialog2	Opens dialog box for editing the advanced Bode plot probe
EditCrosshairDimensionDialog	UI function, opens dialog for editing cursor dimension
EditCurveMarkerDialog	UI function, opens dialog to edit curve marker
EditDeviceDialog	UI function to select device and edit device parameters
EditDigInitDialog	UI function to edit digital initial condition
EditFileDefinedPWLDialo	Opens a dialog box designed for editing the SIMPLIS File Defined piece wise linear (PWL) sources
EditFreeTextDialog	UI function, opens dialog to edit graph free text object
EditGraphMeasurement	Edit an existing graph measurement object.
EditGraphTextBoxDialog	UI function, opens dialog to edit graph text box object
EditJumperDialog	Opens edit jumper dialog
EditLegendBoxDialog	UI function, opens dialog to edit graph legend box object
EditObjectPropertiesDialog	UI function, opens dialog to edit property values
EditPinDialog	UI function to edit symbol editor pins

Function Name	Description
EditPotDialog	UI function to edit potentiometer properties
EditProbeDialog	UI function, opens edit fixed probe dialog
EditPropertyDialog	UI function to edit symbol editor properties
EditReactiveDialog	Opens a dialog box designed to edit inductors and capacitors
EditSelect	User interface function. Returns entries in up to 6 edit controls
EditSimplisLaplaceFilterDialog	Opens dialog box fro editing SIMPLIS Laplace filters
EditSimplisMosfetDriverDialog	Opens a specialized dialog used to edit the parameters for a SIMPLIS Multi-Level MOSFET Driver.
EditStylesDialog	Opens the Edit Styles dialog
EditSymbolBusDialog	Dialog box for editing offset and size of symbol bus
EditTimer	Edit a timer
EditWaveformDialog	Opens the dialog box editing a time domain waveform
EditWaveformStrDialog	Opens the dialog box editing a time domain waveform
ElementProps	Returns selected element's properties
EncodeImageToBase64	Returns the Base64 binary encoding of a png or bmp file
EnterTextDialog	UI function to define multi line text
EpochTime	Returns absolute time in seconds
erf	Calculate erf(x)
erfc	Calculate erfc(x)
EscapeString	Process string and replace escaped characters with literals
EscapeStringEncode	Process string and replace literals with escaped characters.
ev	Special function used to evaluate a sequence of expressions
Execute	Execute script as a function
ExistCommand	Tests if a command exists
ExistDir	Checks if the specified directory exists
ExistFile	Tests whether a file exists
ExistFunction	Returns TRUE if the specified function exists.
ExistSymbol	Returns TRUE if specified schematic symbol exists.
ExistVec	Returns TRUE if specified schematic symbol exists.
exp	Exponential

Function Name	Description
fd	Returns final division of a multi-division vector. Alias of FinalDivision FinalDivision()
fft	Fast Fourier Transform
Field	Provides bit-wise access to integers
FileToString	Returns the entire contents of a file as a single string
FilterEditMenu	Filters a menu list to return only menu definitions that are actually displayed
FilterFile	Filters specific lines from a text file.
FinalDivision	Returns final division of a multi-division vector
FindGraphMeasurement	Returns graph measurement objects that match a provided curve ID and, optionally, a Label value
FindModel	Returns location of simulator model given name and type
FIR	Finite Impulse Response digital filter
Floor	Returns argument truncated to next lowest integer
floorv	Returns arguments truncated to next lowest integers, as a vector
FormatNumber	Returns formatted number in string form
Fourier	Performs a spectral analysis using the continuous Fourier algorithm
FourierOptionsDialog	UI function, opens fourier options dialog
FourierWindow	Apply window function for fourier analysis
FullPath	Returns full path name of given relative path
gamma	Calculate gamma(x)
Gauss	Returns random number with Gaussian distribution
GaussLim	Returns random number with Gaussian distribution truncated at the tolerance limits. Alias of function GaussTrunc()
GaussTrunc	Returns random number with Gaussian distribution truncated at the tolerance limits
GenPrintDialog	UI function, opens print dialog box
GetActualPath	Returns a file system path resolving any links
GetAllAxes	Returns array of axis ids for all axes in current graph.
GetAllCurves	Returns array of curve indexes for all curves in current graph
GetAllSimulatorDevices	Returns details of all simulator built-in devices
GetAllSymbolPropertyNames	Finds names of all the properties on currently open symbol

Function Name	Description
GetAllXAxes	Returns array of axis ids for all x axes in current graph.
GetAllYAxes	Returns array of axis ids for all y axes in current graph.
GetAnalysisInfo	Return information about most recent analysis
GetAnalysisLines	Returns the analysis lines used in the most recent simulation analysis
GetAnnotationText	Returns the text of the requested annotation
GetAxisCurves	Returns array of curve id's for all curves attached to specified axis
GetAxisLimits	Returns min and max limits and axis type (log or lin) of specified axis
GetAxisType	Obsolete - do not use for new code.
GetAxisUnits	Returns units of specified axis
GetChildModulePorts	Finds information about module ports in the underlying schematic of a hierarchical block
GetCodecNames	Returns encoding types available
GetColours	Return names of all colour objects
GetColourSpec	Return specification for a colour object
GetCompatiblePathName	Returns a path name with no white space.
GetComponentValue	Special function to get a component value or parameter
GetConfigLoc	Return location of config information
GetConnectedPins	Returns instance and pin name for all instances connected to net at specified point
GetConvergenceDevNames	List device references that have convergence info available
GetConvergenceInfo	Return convergence data for most recent simulation
GetConvergenceNodeNames	List node names that have convergence info available
GetCurDir	Returns current working directory.
GetCurrentGraph	Returns id of the currently selected graph.
GetCurrentStepValue	Get current step value in a script-based multi-step analysis
GetCursorCurve	Returns curve id and source group name of curve attached to measurement cursor
GetCurveAxes	Returns X and Y axis ids of specified curve
GetCurveAxis	Returns y-axis id of specified curve

Function Name	Description
GetCurveCreationIds	Returns IDs of objects created by the most recent Plot, Curve or NewPlot command.
GetCurveName	Returns the label of specified curve
GetCurves	Returns curve names in selected graph
GetCurveVector	Returns data associated with a graph curve
GetDatumCurve	Returns curve id and source group name of curve attached to reference cursor
GetDeviceDefinition	Retrieve the text of a model definition from library
GetDeviceInfo	Returns information about the specified simulator device
GetDeviceParameterNames	Returns list of device parameter names for specified SPICE device
GetDevicePins	Get electrical connections of a simulator device
GetDeviceStats	Get simulation statistics for each device type
GetDisabledInstances	Returns a list of all disabled instances on the current schematic
GetDivisionLabels	Get the division labels for a specified graph curve
GetDotFilePath	Retrieves the temporary file used to implement a .FILE statement in a netlist
GetDotParamNames	Returns names of .PARAM variables used in latest simulation
GetDotParamValue	Returns value of specified .PARAM value in latest simulation run
GetDriveType	Determines the type of drive or file system of the specified path
GetEmbeddedFileName	Returns the actual file name used for an embedded file specified using '.FILE' and '.ENDF'
GetEnvVar	Return specified system environment variable
GetEthernetAddresses	Returns information about the installed Ethernet adapters
GetF11Lines	Returns the contents of the schematic's text window (also known as the F11 window)
GetFailedNodesFromIterNum	Get list of failed nodes given an iteration value
GetFile	User interface function. Returns user selected file name
GetFileCD	User interface function. As GetFile but changes directory.
GetFileDir	Get the directory where the specified file is located
GetFileExtensions	Returns file extensions for specified SIMetrix file type

Function Name	Description
GetFileInfo	Returns information about a specified file
GetFileSave	User interface function. Returns user selected file name for saving
GetFileVersionStamp	Returns file version stamp
GetFileViewerSelectedDirectories	Returns directory paths of selected directories in the File Views
GetFileViewerSelectedFiles	Returns file names of selected files in the File Views
GetFirstSelectedElementOfType	Returns handle of first selected schematic element of the requested type
GetFonts	Return names of all font objects
GetFontSpec	Return specification for named font
GetFreeDiskSpace	Returns space available on specified disk volume
GetGraphFromWindow	Return graph ID from a Window ID
GetGraphObjects	Return IDs for specified graph objects
GetGraphObjPropNames	Return property names for specified graph object
GetGraphObjPropValue	Return value for a graph object property
GetGraphObjPropValues	Return value for a graph object property
GetGraphTabs	Return graph ids for graph tabbed sheets
GetGraphTitle	Return current graph title
GetGridAxes	Returns ids of all axes in the specified Grid
GetGridCurves	Returns ids of all curves in the specified Grid
GetGroupAnalysisInfo	Retrieves analysis info stored in data groups
GetGroupFromAnalysisId	Get group name given its analysis ID
GetGroupInfo	Returns information about a group
GetGroupStepParameter	Returns the name of the 'stepped parameter' of a multi-step run
GetGroupStepVals	Returns the 'stepped values' in a multistep run
GetGUID	Create a GUID
GetHighlightedWidgetId	Returns ID of highlighted widget
GetHostId	Get hostid that can be used for licensing
GetHttpContentSize	Get the size of an HTTP object
GetInstanceBounds	Returns the bounds occupied by a schematic instance
GetInstanceConvergenceInfo	Returns convergence information about a device.
GetInstanceParamValues	Returns parameter values for a simulator device
GetInstancePinLocs	Returns pin locations of specified instance

Function Name	Description
GetInstsAtPoint	Returns instances at specified point
GetInternalDeviceName	Finds the simulator's internal device name for a model
GetKeyDefs	Returns details of all key definitions created using DefKey
GetKnownFolderPath	Get system path location
GetLaplaceErrorMessage	Convert ParseLaplace() status code to an error message
GetLastCommand	Retrieve last command issued by a menu or toolbar
GetLastError	Returns result of most recent command
GetLastGraphObjectAdded	Returns ID of last graph object added
GetLegendProperties	Returns array of measurement names or values
GetLibraryModels	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
GetLine	Returns a single line from a file.
GetListSelected	Return list of selected elements from the ListSubsetDialog
GetListUnselected	Return list of unselected elements from the ListSubsetDialog
GetLongPathName	Returns long path name for path specified either as a long or short path
GetMaxCores	Return maximum cores available taking account of hardware capability and license
GetMD5	Get MD5 hash for a file
GetMD5String	Get MD5 hash for a string
GetMenuItems	Returns all menu item names in the specified menu
GetModelFiles	Returns a list of currently installed device models.
GetModelLibraryErrors	Returns list of error messages from model library install operations
GetModelName	Returns the model name used by a simulator device
GetModelParameterNames	Returns the names of all real valued parameters of a simulator device model
GetModelParameters	Returns information about a device's model parameters
GetModelParameterValues	Returns the values of all parameters of a simulator model

Function Name	Description
GetModelType	Returns the simulator internal device name given a user model name
GetModifiedStatus	Returns modified status of the specified schematic
GetNamedSymbolPins	Returns the names for all the pins of a symbol or hierarchical component
GetNamedSymbolPropNames	Returns names of all properties defined for a library symbol
GetNamedSymbolPropValue	Returns the value of a property defined for a library symbol
GetNearestNet	Returns information about the schematic net nearest the mouse cursor
GetNextDefaultStyleName	Returns next fully available default style name
GetNodeConvergenceInfo	Returns convergence information about a node.
GetNodeNames	Returns all node names used in most recent simulation
GetNonDefaultOptions	Returns names of all explicit .OPTION settings in the most recent simulation
GetNumCurves	Returns number of curves in a curve group
GetOpenSchematics	Returns the path names of all open schematics
GetOption	Returns the value of an option variable
GetPath	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	Returns internal read-only status of specified schematic
GetRegistryClassesRootKeys	List sub keys under key in registry HKEY_CLASSES_ROOT root
GetSchematicFileVersion	Returns the file version for the schematic
GetSchematicTabs	Returns the IDs of the schematics.
GetSchematicVersion	Returns version information about the current schematic
GetSchemTitle	Returns the title of the current schematic
GetSelectedAnnotationText	Returns the text in the selected annotation
GetSelectedCurves	Returns the curve IDs of the selected curves
GetSelectedGraphAnno	Obsolete - for new code use GetSelectedGraphAnnotations()

Function Name	Description
GetSelectedGraphAnnotations	
GetSelectedGrid	Returns id of selected Grid
GetSelectedStyleNames	Returns the names of the styles used by the selected elements
GetSelectedXAxis	Returns id of selected X-Axis
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	Returns path name of user selected file
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	Returns list of events for most recent simulation
GetSimulatorMode	Returns the simulator mode of the current schematic
GetSimulatorOption	Returns the value of a simulator option as used by the most recent analysis
GetSimulatorOptionInfo	Returns type and default value of a simulator option setting
GetSimulatorOptions	Return list of simulator options
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

Function Name	Description
GetSymbolOrigin	Returns the location of the symbol editor's symbol origin point
GetSymbolPropertyInfo	Returns information about symbol editor symbol properties
GetSymbolPropertyNames	Returns symbol editor symbol property names
GetSymbols	Returns array of available schematic symbols
GetSystemInfo	Returns information about the user's system
GetTempFile	Creates a temporary file name
GetTextEditorText	Returns text in the selected text based editor
GetThreadTimes	Returns the execution times for each device thread for the most recent simulation
GetTimerInfo	Returns information about a timer object
GetTitleBlockInfo	Returns information about the selected schematic title block
GetToolBarDefinition	Returns names of all buttons on a specified toolbar
GetToolButtons	Returns name and description for available tool buttons
GetTouchstoneErrors	Retrieve error messages from a LoadTouchstone() call
GetUncPath	Returns UNC path of specified path
GetURLFromLocalPath	Encodes a local file path as a URL
GetUsedStyles	Get style names used in the current schematic
GetUserFile	Returns path name of user specified file. Supersedes GetFile() , GetFileCD() , and GetFileSave()
GetVecStepParameter	Returns parameter name associated with vector
GetVecStepVals	Returns parameter values associated with vector
GetWidgetInfo	Returns info about open views
GetXAxis	Returns id of selected X-Axis
GraphImageCapture	Opens Graph Image Capture dialog
GraphImageCaptureAdvanced	Advanced functionality of Graph Image Capture.
GraphImageParameter	Returns the parameters saved to the given global location
GraphLimits	Returns x and y limits of selected graph
GroupDelay	Returns group delay of argument
Groups	Returns array of available groups
GuiType	Returns whether a GUI is enabled
Hash	Returns a 'hash' value for the supplied string

Function Name	Description
HashAdd	Add items to a hash table
HashCreate	Create a hash table
HashDelete	Delete a hash table
HashKeys	Return a list of keys entered into a specified hash table
HashRemove	Remove items to a hash table
HashSearch	Search hash table for an item
HasLogSpacing	Determines whether the supplied vector is logarithmically spaced
HasProperty	Determines whether a particular instance possesses a specified property.
HaveFeature	Determines whether a specified license feature is available
HaveInternalClipboardData	Returns the number of items in the specified internal clipboard
HierarchyHighlighting	
HighlightedNets	Returns names for any wholly highlighted net names on the specified schematic
Histogram	Returns histogram of argument
HistoryDepthDialog	Displays history depth dialog box
Iff	Returns a specified value depending on the outcome of a test
IffV	Returns a specified value depending on the outcome of a test
IIR	Infinite Impulse Response digital filter
im	Returns imaginary part of argument
imag	Returns imaginary part of argument
InitRandom	Initialises the random number generator used for SIMPLIS Monte Carlo distribution functions
InputGraph	User Interface function. Input text for graph operation
InputSchem	User Interface function. Input text for schematic operation
Instances	Returns array of instances possessing specified property
InstNets	Returns array of net names for each pin of selected schematic instance
InstNets2	As InstNets but with more advanced features to identify instance
InstPins	Returns array of pin names for each pin of selected schematic instance

Function Name	Description
InstPoints	Returns location and orientation of specified instance
InstProps	Returns names of all properties owned by selected instance
integ	Returns integral of argument
Interp	Interpolates argument to specified number of evenly spaced points
IsComplex	Returns TRUE if argument is complex
IsComponent	Determines whether a schematic instance is a hierarchical component
IsDocumented	Returns whether the script command or function is documented
IsFileOfType	Returns TRUE if the filename given is of the type checked against
IsFullPath	Returns TRUE if the supplied path name is a full absolute path
IsImageFile	Tests if a file type is an image format.
IsModelFile	Determines if a file contains valid electrical models
IsNum	Returns TRUE if argument is numeric (real or complex)
IsOptionMigrateable	Determines if an option variable may be migrated in a version upgrade.
IsSameFile	Compares two paths and returns true (1) if they point to the same file
IsScript	Determines whether the supplied script name can be located
IsStr	Returns TRUE if argument is a string
IsTextEditor	Returns true if selected editor is a text editor
IsTextEditorModified	Returns true if the highlighted text editor is modified.
Join	Joins a string array into a single string on a separator.
JoinStringArray	Concatenates two string arrays to return a single array
JoinVectors	Concatenates two vectors
length	Returns number of elements in vector.
ListDirectory	Returns file names found in a directory matching a supplied wildcard spec
ListSchemProps	Returns the schematic properties
ln	Natural logarithm
LoadFile	Returns the contents of a text file as a vector

Function Name	Description
LoadSensitivityReport	Displays the contents of a sensitivity XML file in a table
LoadTouchstone	Loads a Touchstone network parameter file
Locate	Locates value in a monotonic vector. Returns index.
log	Base 10 logarithm, same as log10()
log10	Base 10 logarithm
LPF	Applies a continuous time single pole low pass filter to input data
mag	Magnitude (same as abs())
magnitude	Magnitude (same as the function abs())
makecomplexvec	Returns a complex vector of specified length whose elements are all initialised to 0.0
MakeDir	Make a directory and result of operation
MakeLogicalPath	Converts a file system path to a symbolic path
MakeString	Create a string array with specified number of elements
makevec	Returns vector of specified length whose elements are all initialised to 0.0. Can also create a multi-division vector
ManageDataGroupsDialog	Open Manage Data Group dialog box
ManageMeasureDialog	Opens dialog box used to manage graph measurements.
MapArray	Creates a real or string array with items at mapped positions
max	Returns max of two vectors
maxidx	Returns index to maximum input value
Maxima	Returns locations of maxima of specified vector
Maximum	Returns most positive value in vector
mean	Returns statistical mean of all values in vector
Mean1	Returns mean of data in given range
MeasureDialog	Opens dialog for specifying graph measurements
MessageBox	Opens a dialog box with a message and user options
Mid	Returns substring of the given string
min	Returns min of two vectors
minidx	Returns index to minimum input value
Minima	Returns locations of minima of specified vector
Minimum	Returns most negative value in vector

Function Name	Description
MkVec	Returns an expression to access simulation vector data
MLRidgeRegressionFit	Regularised linear regression based fit to given data
MLSplineFit	Spline based fit to given data
MLVector	Creates a vector of consecutively increasing values.
ModelLibsChanged	Returns TRUE if any installed model paths have changed
MSWReadHeader	Read header information of an MSW file
Navigate	Returns path name of hierarchical block given root path and full component reference
NearestInst	Cross probe function. Returns nearest schematic instance to cursor
NetName	Cross probe function. Returns the net name of the nearest wire or instance pin.
NetNames	Returns array of all net names in selected schematic
NetWires	Return all wires on specified net
NewPassiveDialog	UI function to select passive component value and parameters
NewValueDialog	General purpose user input function. Opens a user configurable dialog box
norm	Returns argument scaled so that its largest value is unity.
NumberSelectedAnnotations	Returns number of selected annotations
NumDivisions	Returns number of divisions in a vector
NumElems	Returns number of elements in a vector
OpenEchoFile	Redirects the output of the Echo command
OpenFile	Opens a file and returns its handle. This may be used by the Echo command
OpenPDFPrinter	Sets up printing for PDF output
OpenPrinter	Starts a print session
OpenSchem	Opens a schematic and returns value indicating success or otherwise
OpenSchematic	Opens a schematic without displaying it. Returned ID useable by various functions and commands
OptimiserAnalysisLine	Returns optimiser analysis lines
OptimiserApplyFailedRun	Notify optimiser that a simulation run failed
OptimiserApplySpecification	Evaluates measurements for specified optimiser definition

Function Name	Description
OptimiserCloseDef	Delete an optimiser object
OptimiserCreateFromXML	Create an optimiser definition from an XML file
OptimiserCreateFromXMLString	Creates an optimiser definition from an XML string
OptimiserErrorMessage	Returns string showing error message returned by a filed optimiser run
OptimiserFinish	Instruct optimiser runtime session to terminate
OptimiserGetDataObject	Get data for optimiser data object
OptimiserGetDataObjectNames	Returns names of all data objects in an optimiser definition
OptimiserGetInfo	Get optimiser file information
OptimiserGetIteration	Returns data from a single optimiser iteration
OptimiserGetOptions	Returns options for an optimiser definition
OptimiserGetParameters	Returns parameter definition from an optimiser definition
OptimiserGetParameterValues	Gets parameter values for current optimiser iteration
OptimiserGetSpecification	Returns details of all analyses and specifications for an optimiser definition
OptimiserLoadWidgetFromXML	Loads an optimiser widget (GUI) from an XML file
OptimiserParameterLine	Gets parameter line for current optimiser iteration
OptimiserResults	Returns information about the results of an optimiser run
OptimiserRunning	Returns 1 when the optimiser is running
OptimiserSimulatorGetDef	Returns an optimiser ID for the most recent simulator's optimiser run
OptimiserSimulatorGetInfo	Get simulator optimiser file information
OptimiserSimulatorResults	Returns information about the results of a simulator optimiser run
OptimiserSimulatorStatus	Returns current status of the simulator's optimiser
OptimiserSimulatorUserMessage	Returns a string providng the current state of the simulator's optimiser for display to the user
OptimiserSimulatorWriteXMLString	Returns the XML definition of the current state of the simulator's optimiser as a string
OptimiserStart	Start optimiser run
OptimiserStatus	Returns current status of an optimiser
OptimiserSuccess	Returns 1 if optimiser run has completed successfully
OptimiserUserMessage	Returns a string providng the current state of a running optimiser for display to the user
OptimiserWidgetCreateOptDef	Return optimiser definition ID from widget (GUI)

Function Name	Description
OptimiserWidgetGetDataObject	Returns data for an optimiser widget (GUI) data object
OptimiserWidgetGetDataObjectNames	Returns names of all data objects in the current optimiser widget (GUI)
OptimiserWidgetGetInfo	Get optimiser widget (GUI) file information
OptimiserWidgetPreFlightChecks	Check optimiser widget (GUI) data prior to starting an optimiser run
OptimiserWidgetWriteCommandFile	Writes out SIMetrix simulator command file from current widget (GUI)
OptimiserWidgetWriteXML	Writes current optimiser widget to a file
OptimiserWidgetWriteXMLString	Returns string containing XML definition of current optimiser widget (GUI)
OptimiserWriteCommandFile	Create SIMetrix simulator command file from optimiser definition
OptimiserWriteHTMLReport	Generate an HTML report from an optimiser XML results file.
OptimiserWriteXML	Writes out an optimiser definition to a file
OptimiserWriteXMLString	Returns an optimiser XML definition as a string
Parse	Splits up the string supplied as argument 1 into substrings or tokens
ParseAnalysis	Opens the choose analysis dialog
ParseEscape	Splits up the string supplied as argument 1 into substrings or tokens with escaped delimiters, eliminating empty tokens
ParseLaplace	Parses a Laplace expression to return array of denominator and numerator coefficients
ParseParameterString	Legacy function. Use ParseParameterString2() for new code. Parses a string of name-value pairs and performs some specified action on them.
ParseParameterString2	Parses a string of name-value pairs and performs some specified action on them
ParseProbeExpression	Parses an expression used arbitrary fixed probes
ParseSIMPLISInit	Reads and parses the .init file created by a SIMPLIS run
ParseSpiceLines	Parses a string in SPICE format and returns each line in a string array
PathEqual	Compares two path names with platform dependent case-sensitivity
PerCycleTiming	Returns a vector of “Per Cycle” Frequency, Period, Duty Cycle, On-Time, or Off-Time values.

Function Name	Description
PerCycleValue	Returns a vector of "Per Cycle" Minimum, Maximum, Mean, Peak-to-Peak, or RMS values.
ph	Returns phase of argument in degrees
phase	Returns phase of argument in degrees
phase_rad	Returns phase of argument in radians
PhysType	Returns the physical type of the argument
PinName	Cross probe function. Returns pin name nearest to cursor
PrepareSetComponentValue	Configures SetComponentValue() function
PreProcessNetlist	Preprocess netlist
Probe	Displays probe cursor in schematic and waits for mouse click
ProcessingAccelerator	Detects if the current script was called by an accelerator key
ProcessingDragAndDrop	Detects if the current script was called by a drag and drop operation
ProcessingGuiAction	Detects if the current script was called by a GUI action
Progress	UI function, opens progress bar
PropFlags	Returns the attribute flags of a schematic property
PropFlags2	Returns the attribute flags of a schematic property
PropFlagsAll	Returns selected property flags for all selected elements with optional filtering
PropFlagsAnnotations	Returns selected property flags for all selected annotations with optional filtering
PropFlagsWires	Returns selected property flags for all selected wires with optional filtering
PropOverrideStyle	Returns override style of selected property
PropValue	Returns value of specified property for selected instance
PropValues	Returns array of property values
PropValues2	As PropValues but with rearranged arguments
PropValuesAll	Returns selected property values for all selected elements with optional filtering
PropValuesAnnotations	Returns selected property values for all selected annotations with optional filtering
PropValuesWires	Returns selected property values for all selected wires with optional filtering
PutEnvVar	Write an environment variable

Function Name	Description
PWLCurveFit	Returns a vector with a Piecewise Linear Approximation of the input vector.
PWLDialog	Opens a dialog box designed for editing piece wise linear sources
QueryData	Filters a list of data items according to search criteria
RadioSelect	User interface function. Returns user selection of up to 5 radio buttons
Range	Returns range of vector (accepts, real, complex and string)
re	Returns real part of argument
ReadClipboard	Returns text contents of the windows clipboard
ReadConfigCollection	Returns the contents of an entire section in the configuration file
ReadConfigSetting	Reads a configuration setting
ReadF11Analyses	Read analysis specification in schematic F11 window
ReadF11Options	Read .OPTIONS line in the F11 window
ReadFile	Reads text file and returns contents as an array of strings
ReadIniKey	Reads a key in an "INI" file
ReadRegSetting	Reads a string setting from the windows registry
ReadSchemProp	Returns value of schematic window property value.
ReadSIMPLISF11Data	Returns information about the current SIMPLIS simulation configuration
ReadSpiceFile	Reads text file and returns contents as an array of strings. Identifies SPICE netlist continuation lines and inline comments
ReadTextEditorProp	Reads a text editor property
ReadTouchstone	Retrieves the data from a previously loaded Touchstone file.
real	Returns real part of argument
Ref	Returns reference of argument
RefName	Returns the name of the arguments reference vector
RegexContains	Searches a string for a regular expression match and returns 1 or 0 indicating whether the regular expression matches the string.
RegexIsValid	Checks the regular expression string provided at the first index for validity.
RegexMatch	Searches an input string for a regular expression and returns the matching strings as a string array.

Function Name	Description
RegExp	Match an input string to a sequence of regular expressions
RegExpReplace	Searches an input string for a regular expression and returns the input string with the regular expression replaced.
RegExpSearch	Searches an input string array for a regular expression match and returns the index where the first match occurs.
RegExpSearchAll	Searches an input string array for a regular expression match and returns the indexes where the matches occur.
RegExpSplit	Splits a input string into tokens defined by a regular expression.
RegExpStrStr	Searches an input string for a regular expression match and returns the index into the input string where the match occurs.
RelativePath	Returns a relative path name given a full path and a reference path
RemapDevice	Map SIMetrix simulator device to model name and level number
RemoveConfigCollection	Removes one or more entries from a configuration file collection
RemoveModelFile	Uninstalls a model path
RemoveSymbolFiles	Removes a symbol file or set of symbol files from the symbol library
ResolveGraphTemplate	Evaluate template string used by graph object
ResolveTemplate	Evaluate template string
RestartTranDialog	UI function, opens restart transient dialog
Rms	Returns accumulative RMS value of argument
RMS1	Returns RMS of argument over specified range
rnd	Returns random number
RootSumOfSquares	Returns root sum of squares of argument over specified range
rt	Evaluate template string
SaveSpecialDialog	Opens the dialog used by the schematic's Save Special... menu
Scan	Splits a character delimited string into its components.
ScanEscape	Splits up the string supplied as argument 1 into substrings or tokens with escaped delimiters, preserving empty tokens

Function Name	Description
ScriptName	Return name of currently executing script
Search	Search for a string in a list of strings
SearchModels	Special purpose used by library installation. Returns pathnames of SPICE compatible model files
Seconds	Returns the number of seconds elapsed since January 1, 1970
Select2Dialog	Displays a dialog offering two lists
SelectAnalysis	Opens choose analysis dialog box. Returns value according to how box closed
SelectColourDialog	UI function, opens colour selection dialog
SelectColumns	Analyses an array of character delimited strings and returns selected values.
SelectCount	Returns number of selected items on schematic
SelectDevice	Special function forms part of parts browser system. Takes catalog data as arguments and opens dialog box to select a device.
SelectDialog	User interface function. Allows selection of one or more items from list
SelectedProperties	Returns information about selected properties
SelectedStyleInfo	Returns style information for the selected element
SelectedWires	Returns handles to selected wires on schematic
SelectFontDialog	UI function, opens select font dialog
SelectRows	Analyses an array of character delimited strings and returns selected values.
SelectSIMPLISAnalysis	Opens SIMPLIS choose analysis dialog box
SelectSymbolDialog	Opens a dialog box allowing the user to select a schematic symbol from the symbol library
SelGraph	Returns id of selected graph.
SelSchem	Returns TRUE if at least one schematic is open.
SetComponentValue	Special function to set or get a component value or parameter
SetDifference	Finds the difference between two sets of strings
SetInstanceParamValue	Set an instance parameter during a script-based multi-step analysis
SetIntersect	Finds the intersect of two sets of strings
SetModelParamValue	Set a model parameter during a script-based multi-step analysis
SetPropertyStyles	Sets styles as property styles

Function Name	Description
SetReadOnlyStatus	Sets read-only/writeable status of specified schematic
SetSymmetricDifference	Finds the symmetric difference between two sets of strings
SetUnion	Finds the union of two sets of strings
Shell	Runs an external program and returns its exit code
ShellExecute	Performs an operation on a windows registered file
ShiftRef	Returns a vector with the reference values shifted by a specified amount, or if the second argument is not provided, shifts the reference values so the first data point has a reference value of zero.
sign	Returns sign of argument
SimetrixFileInfo	Returns information about a SIMetrix file
SIMPLISRunStatus	Tests if a SIMPLIS simulation is running
SIMPLISSearchIdx	Searches input string array for a test string, returning the indices into input array array where the test string matches.
SimulationHasErrors	Determines success of most recent simulation
sin	Sine (radians)
sin_deg	Sine (degrees)
sinh	Hyperbolic sine (radians)
Sleep	Executes a timed delay
Sort	Performs alphanumeric sort on argument.
SortIdx	Deprecated, use SortIdx2() for new code
SortIdx2	Sorts any vector and returns index order
SourceDialog	User Interface function. Opens source dialog box for specifying of voltage and current source. Returns string with user selected values
SpectrumUniv	General purpose function performs a Fourier analysis on a vector
SplitPath	Splits file system path into its components
SplitString	Splits string into parts according to single token
SprintfNumber	Returns a formatted string
sqrt	Square root
Str	Converts argument to string
StringLength	Returns the number of characters in the supplied string.
StringStartsWith	Checks whether a string starts with another string.
StrStr	Locates a sub string within a string

Function Name	Description
StyleInfo	Returns style information
StyleLineTypes	Returns list of possible style line types
StyleNames	Returns a list of style names
SubstChar	Substitutes characters in string
SubstProbeExpression	Substitutes node names in a probe expression. Used by fixed probe symbol
SubstString	Replaces a substring in a string, case sensitive
sum	Sums the arguments
SumNoise	Returns root sum of squares of argument over specified range
SupportedReadFormats	Returns names of image formats types supported for display in SIMetrix windows such as schematics.
SupportedWriteFormats	Returns names of image formats types that SIMetrix can create for graphical windows such as schematics and graphs.
SxUUID	Produces universally unique identifiers.
SymbolInfoDialog	Returns name of schematic symbol
SymbolLibraryManagerDialog	Opens the Symbol Library Manager dialog box
SymbolName	Returns symbol name of specified instance
SymbolNames	Returns symbol names of schematic instances
SymbolPinOrder	Set and/or return pin order of symbol editor symbol
SymbolPinPoints	Returns the location of specified pin
SystemValue	Returns the value of a system defined variable
SystemValuePath	Returns the value of a system defined variable that accesses a file system path
SystemWidgetExistsInSelectedWindow	States whether particular system view is in the highlighted window
TableDialog	Displays a spreadsheet style table to allow the user to enter tabular data
TableEditor	Displays a table of combo boxes to allow select tabular data
tan	Tangent (radians)
tan_deg	Tangent (degrees)
tanh	Hyperbolic tangent
TemplateGetPropValue	Function returns the value of a property. For use in template scripts only
TemplateResolve	Resolve TEMPLATE value. For use in template scripts only

Function Name	Description
TextEditorHasComments	Returns whether the editor supports comments
ThdWeight	Returns a vector of weighting coefficients used to weight the harmonic coefficients before making a THD measurement.
TickCount	Returns a time in seconds suitable for timing measurement
Time	Return system time as string
ToLower	Converts a string to all lower case
ToUpper	Converts a string to all upper case
TranslateLogicalPath	Converts symbolic path to a physical path
TreeListDialog	General purpose UI function. Open dialog box with tree list control
True	Returns 1 if vector exists and is nonzero
Truncate	Returns vector that is a sub range of supplied vector
TwoFileSelectionDialog	General purpose file dialog with two file entries and an option third description entry
UD	Alias of Distribution()
UngroupCurve	Ungroup a multi-step curve
Unif	Returns random number with uniform distribution
Units	Returns physical units of argument
unitvec	Returns vector of specified length whose elements are all 1
UpDownDialog	General purpose UI function. Opens dialog with up-down list to allow rearranging order
UserParametersDialog	UI function, opens dialog allowing editing of user parameter values
Val	Converts argument to value
ValueDialog	User interface function. Opens dialog with up to 10 boxes for entering numeric values. Return array of user selected values
Vec	Returns data for named vector. (Allows access to vectors with invalid names)
vector	Returns vector of specified length with each element equal to its index
VectorsInGroup	Returns array of variable names belonging to specified group
VersionInfo	Returns version information about running copy of SIMetrix
ViewFormattedFile	View HTML formatted text read from a file

Function Name	Description
ViewFormattedText	View HTML formatted text
WAV_CloseFile	Close WAV file
WAV_OpenFile	Open a WAV file
WAV_ReadData	Read WAV data from file
WAV_WriteFile	Write data to a file in WAV format
WC	Returns random number with worst case distribution
WC2	Returns random number with worst case distribution with asymmetric tolerances.
WirePoints	Returns location of specified wire
Wires	Return all wires in schematic
WM_CanRevertToSaved	Returns whether chosen editor has a revertable saved state.
WM_GetAllVisibleContentWidgetNames	Returns visible content windows of a given type in all windows
WM_GetCentralWidgetGeometry	Returns window geometry information
WM_GetContentWidgetNames	Returns content widget names
WM_GetContentWidgetSessionInfo	Returns widget session information
WM_GetContentWidgetsLayout	Returns layout information
WM_GetContentWidgetsOfType	Returns content windows of a given type, either in a specific main window or in all windows
WM_GetContentWidgetTypes	Returns the workspace view types in a particular window
WM_GetCurrentWidgetInfo	Returns info of most recently accessed widget (GUI window) of given type
WM_GetCurrentWindowName	Returns name of highlighted window
WM_GetLastAccessedContentWidget	Return window name of last accessed tab sheet.
WM_GetNumberModifiedEditors	Returns number of editors that are modified in all windows
WM_GetPrimaryWindowName	Returns the name of the primary window
WM_GetSystemWidgetSessionInfo	Returns widget session information
WM_GetSystemWidgetsLayout	Returns layout information
WM_GetWidgetInstanceInfo	Returns information about the specified widget (GUI Window)
WM_GetWindowGeometry	Returns window geometry
WM_GetWindowNames	Returns the names of all windows
WM_NumberContentWidgets	Returns the number of content widgets in use
WM_NumberSystemWidgets	Returns the number of system widgets in use

Function Name	Description
WriteConfigSetting	Writes a configuration setting
WriteF11Lines	Writes lines directly to the F11 window overwriting any existing lines
WriteF11Options	Write SIMetrix simulator options to the F11 window.
WriteIniKey	Writes a key value to an 'INI' file
WriteRawData	Writes data to the specified file in a SPICE3 raw file compatible format
WriteRegSetting	Writes a string value to the windows registry
WriteSchemProp	Write schematic window property value
XCursor	Returns x location of graph cursor
XDatum	Returns x location of graph reference cursor
XFromY	Returns array of values specifying horizontal locations where specified vector crosses given y value
XMLCountElements	Returns the number of elements of a particular type
XMLGetAttribute	Returns the attribute value for given name at the current focus element
XMLGetElements	Lists elements at the current focus level
XMLGetText	Returns the text for the current focus element
XMLToString	Returns the XML document as a string
XY	Creates an XY Vector from two separate vectors
XYOrdered	Creates an XY Vector from two separate vectors. Arranges the data points in X order.
YCursor	Returns y location of graph cursor
YDatum	Returns y location of graph reference cursor
YFromX	Returns array of values specifying the vertical value of the specified vector at the given x value

3.1 Functions by Application

3.1.1 Configuration/Licensing

AddConfigCollection	GetFontSpec	ReadConfigCollection
DeleteConfigCollection	GetLicenseInfo	ReadConfigSetting
GetColours	GetLicenseStats	RemoveConfigCollection
GetColourSpec	GetOption	VersionInfo
GetConfigLoc	GetSimConfigLoc	WriteConfigSetting
GetFileExtensions	HaveFeature	
GetFonts	IsOptionMigrateable	

3.1.2 Data fitting

GraphImageCapture	GraphImageParameter	MLVector
	MLRidgeRegressionFit	PWLCurveFit
GraphImageCaptureAdvanced	MLSplineFit	

3.1.3 Dialogs

ACSourceDialog	DialogDesigner	GenPrintDialog
ACSourceDialogStr	EditArcDialog	GetFile
AddPropertyDialog	EditAxisDialog	GetFileCD
AddRemoveDialog	EditBodePlotProbeDialog	GetFileSave
AddRemoveDialogNew		GetSimetrixFile
BoolSelect	EditCrosshairDimensionDialog	GetUserFile
ChooseDir	EditCurveMarkerDialog	InputGraph
CreateDiodeDialog	EditDeviceDialog	InputSchem
	EditDigInitDialog	ManageDataGroupsDialog
CreateNewTitleBlockDialog		ManageMeasureDialog
CurveEditDialog	EditFileDefinedPWLDialg	MeasureDialog
DCSourceDialog	EditFreeTextDialog	MessageBox
DCSourceDialogStr	EditGraphTextBoxDialog	NewPassiveDialog
DefineADCDialg	EditLegendBoxDialog	NewValueDialog
DefineArbSourceDialog	EditObjectPropertiesDialog	Progress
DefineBusPlotDialog	EditPinDialog	PWLDialg
DefineCounterDialog	EditPotDialog	RadioSelect
DefineCurveDialog	EditProbeDialog	RestartTranDialog
DefineDACDialog	EditPropertyDialog	SaveSpecialDialog
DefineFourierDialog	EditReactiveDialog	Select2Dialog
DefineFourierProbeDialog	EditSelect	SelectColourDialog
DefineIdealTxDialog		SelectDevice
DefineLaplaceDialog	EditSimplisLaplaceFilterDialog	SelectDialog
DefineLogicGateDialog		SelectFontDialog
DefinePerfAnalysisDialog	EditSimplisMosfetDriverDialog	SelectSymbolDialog
DefineRegisterDialog	EditStylesDialog	SourceDialog
DefineRipperDialog	EditSymbolBusDialog	SymbolInfoDialog
DefineSaturableTxDialog	EditWaveformDialog	
DefineShiftRegDialog	EditWaveformStrDialog	SymbolLibraryManagerDialog
DefineSimplisMultiStepDialog	EnterTextDialog	TableDialog
	FourierOptionsDialog	

TableEditor	UpDownDialog	ViewFormattedFile
TreeListDialog	UserParametersDialog	ViewFormattedText
TwoFileSelectionDialog	ValueDialog	

3.1.4 File/Directory

CanOpenFile	FullPath	IsFileOfType
ChangeDir	GetActualPath	IsFullPath
ChooseDirectory	GetCodecNames	IsSameFile
CloseEchoFile	GetCompatiblePathName	ListDirectory
CloseFile	GetCurDir	LoadFile
ConvertLocalToUnix	GetDriveType	LoadTouchstone
ConvertUnixToLocal	GetFileDir	MakeDir
CopyTree	GetFileInfo	MakeLogicalPath
CopyURL	GetFileVersionStamp	OpenEchoFile
CreateLockFile	GetFreeDiskSpace	OpenFile
CreateShortcut	GetHttpContentSize	ReadFile
DeleteTouchstone	GetLine	ReadSpiceFile
DeleteTree	GetLongPathName	ReadTouchstone
DeleteTreeProgress	GetPath	RelativePath
DescendDirectories	GetShortPathName	SimetrixFileInfo
DirectoryIsWriteable	GetTempFile	SplitPath
ExistDir	GetTouchstoneErrors	TranslateLogicalPath
ExistFile	GetUncPath	

3.1.5 Graph

AddGraphCrossHair	GetAxisLimits	GetDatumCurve
CreateGraphMeasurement	GetAxisType	GetDivisionLabels
CreateSharedAxisConnector	GetAxisUnits	GetGraphFromWindow
cv	GetCurrentGraph	GetGraphObjects
EditGraphMeasurement	GetCursorCurve	GetGraphObjPropNames
FindGraphMeasurement	GetCurveAxes	GetGraphObjPropValue
GetAllAxes	GetCurveAxis	GetGraphObjPropValues
GetAllCurves	GetCurveCreationIds	GetGraphTabs
GetAllXAxes	GetCurveName	GetGraphTitle
GetAllYAxes	GetCurves	GetGridAxes
GetAxisCurves	GetCurveVector	GetGridCurves

GetLastGraphObjectAdded	GetSelectedGrid	SelGraph
GetLegendProperties	GetSelectedXAxis	UngroupCurve
GetNumCurves	GetSelectedYAxis	XCursor
GetSelectedCurves	GetXAxis	XDatum
GetSelectedGraphAnno	GraphLimits	YCursor
GetSelectedGraphAnnotations	ResolveGraphTemplate	YDatum

3.1.6 Mathematical

abs	Fourier	min
acos	FourierWindow	minidx
acosh	gamma	Minima
area	GroupDelay	Minimum
arg	HasLogSpacing	norm
arg_rad	Histogram	ph
asin	Iff	phase
asinh	IffV	phase_rad
atan	IIR	re
atan2	im	real
atan2_deg	imag	Rms
atan_deg	integ	RMS1
atanh	Interp	rnd
avg	ln	RootSumOfSquares
cos	log	sign
cos_deg	log10	sin
cosh	LPF	sin_deg
db	mag	sinh
diff	magnitude	SpectrumUniv
erf	makecomplexvec	sqrt
erfc	makevec	sum
exp	max	SumNoise
fft	maxidx	tan
Field	Maxima	tan_deg
FIR	Maximum	tanh
Floor	mean	ThdWeight
floorv	Mean1	unitvec

3.1.7 Miscellaneous

EncodeImageToBase64	HaveInternalClipboardData	ParseLaplace
GetLaplaceErrorMessage	IsImageFile	SupportedReadFormats
		SupportedWriteFormats

3.1.8 Model Library

AddModelFiles	GetModelFiles	RemoveModelFile
AssociateModel	GetModelLibraryErrors	SearchModels
FindModel	IsModelFile	
GetLibraryModels	ModelLibsChanged	

3.1.9 Monte Carlo Distribution

Distribution	GaussTrunc	Unif
Gauss	InitRandom	WC
GaussLim	UD	WC2

3.1.10 Monte Carlo/Sensitivity/Worst-case support

AppendSensitivityData	BuildSensitivityHTML	LoadSensitivityReport
BuildMcolgHTML	BuildWorstCaseHTML	MSWReadHeader

3.1.11 Optimiser

OptimiserAnalysisLine	OptimiserGetIteration	OptimiserSimulatorStatus
OptimiserApplyFailedRun	OptimiserGetOptions	OptimiserSimulatorUserMessage
	OptimiserGetParameters	
OptimiserApplySpecification		OptimiserSimulatorWriteXMLString
OptimiserCloseDef	OptimiserGetParameterValues	OptimiserStart
OptimiserCreateFromXML	OptimiserGetSpecification	OptimiserStatus
		OptimiserSuccess
OptimiserCreateFromXMLString	OptimiserLoadWidgetFromXML	OptimiserUserMessage
OptimiserErrorMessage	OptimiserParameterLine	
OptimiserFinish	OptimiserResults	OptimiserWidgetCreateOptDef
OptimiserGetDataObject	OptimiserRunning	
	OptimiserSimulatorGetDef	OptimiserWidgetGetDataObject
OptimiserGetDataObjectNames	OptimiserSimulatorGetInfo	
OptimiserGetInfo	OptimiserSimulatorResults	OptimiserWidgetGetDataObjectNames

OptimiserWidgetGetInfo	OptimiserWidgetWriteXML	OptimiserWriteHTMLReport
OptimiserWidgetPreFlightChecks	OptimiserWidgetWriteXMLString	OptimiserWriteXML
OptimiserWidgetWriteCommandFile	OptimiserWriteCommandFile	OptimiserWriteXMLString

3.1.12 SIMPLIS

GetSIMPLISExitCode	SelectSIMPLISAnalysis
ParseSIMPLISInit	SIMPLISRunStatus

3.1.13 Schematic

Branch	GetSchemTitle	PrepareSetComponentValue
CloseSchematic		Probe
CloseSchematicTab	GetSelectedAnnotationText	PropFlags
DelSchemProp	GetTitleBlockInfo	PropFlags2
DescendHierarchy	HasProperty	PropFlagsAll
ElementProps	HierarchyHighlighting	PropFlagsAnnotations
GetAnnotationText	HighlightedNets	PropFlagsWires
GetChildModulePorts	Instances	PropValue
GetComponentValue	InstNets	PropValues
GetConnectedPins	InstNets2	PropValues2
GetDisabledInstances	InstPins	PropValuesAll
GetF11Lines	InstPoints	PropValuesAnnotations
GetFirstSelectedElementOfType	InstProps	PropValuesWires
GetInstanceBounds	IsComponent	ReadF11Analyses
GetInstancePinLocs	ListSchemProps	ReadF11Options
GetInstsAtPoint	Navigate	ReadSchemProp
GetListSelected	NearestInst	SelectCount
GetListUnselected	NetName	SelectedProperties
GetModifiedStatus	NetNames	SelectedWires
GetNearestNet	NetWires	SelSchem
GetOpenSchematics		SetComponentValue
GetReadOnlyStatus	NumberSelectedAnnotations	SetReadOnlyStatus
GetSchematicFileVersion	OpenSchem	TemplateGetPropValue
GetSchematicTabs	OpenSchematic	TemplateResolve
GetSchematicVersion	PinName	WirePoints

Wires	WriteF11Options
WriteF11Lines	WriteSchemProp

3.1.14 Schematic Styles

GetNextDefaultStyleName	PropOverrideStyle	StyleInfo
GetSelectedStyleNames	SelectedStyleInfo	StyleLineTypes
GetUsedStyles	SetPropertyStyles	StyleNames

3.1.15 Schematic Symbols and Library

AddSymbolFiles	GetNamedSymbolPropValue	GetSymbolPropertyNames
CompareSymbols	GetSymbolArcInfo	GetSymbols
ExistSymbol	GetSymbolFiles	RemoveSymbolFiles
GetAllSymbolPropertyNames	GetSymbolInfo	SymbolName
GetNamedSymbolPins	GetSymbolOrigin	SymbolNames
GetNamedSymbolPropNames	GetSymbolPropertyInfo	SymbolPinOrder
		SymbolPinPoints

3.1.16 Script

CommandStatus	GetLastError	ScriptName
Execute	IsDocumented	
ExistCommand	IsScript	

3.1.17 Simulator

GetAllSimulatorDevices	GetDevicePins	GetModelName
GetAnalysisInfo	GetDeviceStats	GetModelParameterNames
GetAnalysisLines	GetDotFilePath	GetModelParameters
GetConvergenceDevNames	GetDotParamNames	GetModelParameterValues
GetConvergenceInfo	GetDotParamValue	GetModelType
GetConvergenceNodeNames	GetEmbeddedFileName	GetNodeConvergenceInfo
GetCurrentStepValue	GetFailedNodesFromIterNum	GetNodeNames
GetDeviceDefinition	GetInstanceConvergenceInfo	GetNonDefaultOptions
GetDeviceInfo	GetInstanceParamValues	GetPrintValues
GetDeviceParameterNames	GetInternalDeviceName	GetSimulationErrors
		GetSimulationInfo
		GetSimulationSeeds

GetSimulatorEvents	GetSoaDefinitions	RemapDevice
GetSimulatorMode	GetSoaMaxMinResults	SelectAnalysis
GetSimulatorOption	GetSoaOverloadResults	SetInstanceParamValue
GetSimulatorOptionInfo	GetSoaResults	SetModelParamValue
GetSimulatorOptions	GetThreadTimes	SimulationHasErrors
GetSimulatorStats	ParseAnalysis	
GetSimulatorStatus	PreProcessNetlist	

3.1.18 String

Ascii	HashDelete	rt
BuildParameterString	HashKeys	Scan
Char	HashRemove	ScanEscape
CheckLaplaceExpression	HashSearch	Search
Chr	Join	SelectColumns
ConvertFromBase64	JoinStringArray	SelectRows
ConvertHTMLcolourToRGB	MakeString	SetDifference
ConvertNumberFromBase64	Mid	SetIntersect
ConvertNumberToBase64	Parse	SetSymmetricDifference
ConvertRGBcolourToHTML	ParseEscape	SetUnion
ConvertToBase64	ParseParameterString	SIMPLISSearchIdx
CountChars	ParseParameterString2	Sort
EscapeString	ParseProbeExpression	SortIdx
EscapeStringEncode	ParseSpiceLines	SortIdx2
FileToString	PathEqual	SplitString
FilterFile	QueryData	SprintfNumber
FormatNumber	RegExContains	StringLength
GetGUID	RegExIs Valid	StringStartsWith
GetMD5	RegExMatch	StrStr
GetMD5String	RegExp	SubstChar
GetURLFromLocalPath	RegExReplace	SubstProbeExpression
Hash	RegExSearch	SubstString
HashAdd	RegExSearchAll	ToLower
HashCreate	RegExSplit	ToUpper
	RegExStrStr	
	ResolveTemplate	

3.1.19 System

ConvertIsoTimeToUnix	GetPlatformFeatures	Seconds
ConvertUnixTimeToIso	GetPrinterInfo	Shell
CreateTimer		ShellExecute
Date	GetRegistryClassesRootKeys	Sleep
DeleteTimer	GetSystemInfo	SxUUID
EditTimer	GetTimerInfo	SystemValue
EpochTime	OpenPDFPrinter	SystemValuePath
GetEnvVar	OpenPrinter	TickCount
GetEthernetAddresses	PutEnvVar	Time
GetHostId	ReadClipboard	WriteIniKey
GetKnownFolderPath	ReadIniKey	WriteRegSetting
GetMaxCores	ReadRegSetting	

3.1.20 Text Editor

GetTextEditorText	IsTextEditorModified	TextEditorHasComments
-------------------	----------------------	-----------------------

3.1.21 User Interface

EditBodePlotProbeDialog2	ProcessingGuiAction	WM_GetCurrentWindowName
EditJumperDialog	ReadSIMPLISF11Data	
FilterEditMenu	ReadTextEditorProp	WM_GetLastAccessedContentWidget
GetFileViewerSelectedDirectories	SystemWidgetExistsInSelectedWindow	WM_GetNumberModifiedEditors
GetFileViewerSelectedFiles	WM_CanRevertToSaved	
GetHighlightedWidgetId	WM_GetAllVisibleContentWidgetNames	WM_GetPrimaryWindowName
GetKeyDefs	WM_GetCentralWidgetGeometry	WM_GetSystemWidgetSessionInfo
GetLastCommand		
GetMenuItems	WM_GetContentWidgetNames	WM_GetSystemWidgetsLayout
GetToolBarDefinition		
GetToolButtons	WM_GetContentWidgetSessionInfo	WM_GetWidgetInstanceInfo
GetWidgetInfo	WM_GetContentWidgetsLayout	WM_GetWindowGeometry
GuiType		WM_GetWindowNames
HistoryDepthDialog	WM_GetContentWidgetsOfType	
IsTextEditor	WM_GetContentWidgetTypes	WM_NumberContentWidgets
ProcessingAccelerator		
ProcessingDragAndDrop	WM_GetCurrentWidgetInfo	WM_NumberSystemWidgets

3.1.22 Vector

CurveFit	CurveFitLogX
CurveFitLog	CurveFitLogY

3.1.23 Vectors/Groups

Coll	IsComplex	True
CollateVectors	IsNum	Truncate
ComposeDigital	IsStr	Units
CopyDivisionData	Join Vectors	Val
CyclePeriod	length	Vec
ev	Locate	vector
ExistFunction	MapArray	VectorsInGroup
ExistVec	MkVec	WAV_CloseFile
fd	NumDivisions	WAV_OpenFile
FinalDivision	NumElems	WAV_ReadData
GetGroupAnalysisInfo	PerCycleTiming	WAV_WriteFile
GetGroupFromAnalysisId	PerCycleValue	WriteRawData
GetGroupInfo	PhysType	XFromY
GetGroupStepParameter	Range	XY
GetGroupStepVals	Ref	XYOrdered
GetVecStepParameter	RefName	YFromX
GetVecStepVals	ShiftRef	
Groups	Str	

3.1.24 XML Processing

XMLCountElements	XMLGetElements	XMLToString
XMLGetAttribute	XMLGetText	

Chapter 4

Function Reference

4.1 abs

Returns absolute value or magnitude of argument. This function is identical to the function [mag\(\)](#).

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real array

Absolute value of input

4.2 acos

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Return the inverse cosine of the argument in radians.

4.3 acosh

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Return the inverse hyperbolic cosine of the argument.

4.4 ACSourceDialog

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Initial value

Returns

Return type: Real array

4.5 ACSourceDialogStr

Displays dialog box intended for the user definition of an AC source. Argument is a string array with two elements which specify the initial values for the two controls as follows:

- 0 Magnitude
- 1 Phase

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

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Initial value

Returns

Return type: String array

4.6 AddConfigCollection

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Section name
2	string array	Yes		List of entries
3	string	No		Options

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.

Argument 3

Set to 'nonpath' if the values being stored are not path names. Set to 'usellogicalpaths' if the values being stored are pathnames and they should be stored using logical symbols if possible.

Returns

Return type: real

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.

4.7 AddGraphCrossHair

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](#).

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		curve id

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.

Returns

Return type: string array

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

4.8 AddModelFiles

Installs a list of new models to the model library. Models may be either single files or wildcard specifications. Duplicates will be ignored

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Model path names

Argument 1

String array containing library specifications to be added. A library specification can either be a single file or a wildcard definition, e.g. path*.lb

Returns

Return type: real

Number of models actually installed. This may be less than the number supplied if any are already installed

4.9 AddPropertyDialog

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 in the Prop command](#).

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	<<empty>>	initial property name
2	string	No	0	initial property attribute flags
3	string	No	<<empty>>	initial property value
4	string	No	<<empty>>	options
5	string	No	<<empty>>	font override style
6	string	No	<<empty>>	available font override styles

Returns

Return type: string array length 3

String array of length 4 providing the users settings. The function returns an empty vector if Cancel is selected.

Index	Description
0	Property name
1	Flags value
2	Property value
3	Font override style

4.10 AddRemoveDialog

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		
2	string array	Yes		All items available
3	string array	No	<<empty>>	Options
4	string	No	'horizontal'	Box style

Argument 1

Initial contents of selected list box

Argument 3

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.

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.

Returns

Return type: string array

The function returns the contents of the selected list or an empty vector if "Cancel" is selected. The function will also return an empty vector if there are no selected items and thus it is not possible to use this function to select no items at all. Instead use [AddRemoveDialogNew\(\)](#) if it is necessary to be able to select no items.

4.11 AddRemoveDialogNew

Opens a dialog box to allow user to select from a number of items. This function is identical to [AddRemoveDialog\(\)](#) except that the return value has an additional element to specify the number of selected items. This makes it possible for the selected items list to be empty.

Arguments

No arguments

Returns

Return type: string array

The first element of the result returns the number of items in the selected list which can be zero. This is followed by the items themselves. The return value will an empty vector if "Cancel" is selected.

4.12 AddSymbolFiles

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

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Files to add

Argument 1

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

Returns

Return type: Real

Number of files actually added to the library. This may not be the same length as the argument as the function will not install files that are already installed.

4.13 AppendSensitivityData

Appends new data to an existing sensitivity XML file.

Every sensitivity analysis generates an XML file that contains the results of evaluating the goal functions for each sensitivity case. If the data for the sensitivity analysis is still available, it is possible to add the results of further goal functions to the XML file after the sensitivity analysis has completed. This makes it possible to perform worst-case analyses using goal functions that were not specified at the start of the sensitivity analysis.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Sensitivity XML file to process
2	String			Expression text and sensitivity id
3	Real array	Yes		Values to add to file
4	Real array	Yes		Case numbers

Argument 1

Sensitivity XML file to process

Argument 2

Two element array. The first element provides the text for the expression evaluated. Note that this is used as a label and is not evaluated. The second element is the sensitivity id. If the resulting XML file is to be used as the input to a worst-case analysis, the sensid parameter on the worst-case analysis line must match the sensitivity id provided here.

Argument 3

Array of values to add to XML file

Argument 4

Array of values providing case numbers. The number of items must match the number of values entered in argument 3

Returns

Return type:

Status: success: 1.0, fail: 0.0

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

4.14 area

Calculates the area under the curve of the argument.

This function returns a single value that can be used for measurements. The `integ()` function may be used to obtain a vector of the area. `area(arg)` is equivalent to the value of `(integ(arg))[length(arg)-1]`

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Vector
2	real	No	0.0	Start x value
3	real	No	Final x value in data	End x value

Argument 1

Vector to process. Must have a reference - e.g. x-values

Argument 2

Value on x-axis where the start of the curve area is located

Argument 3

Value on x-axis where the end of the curve area is located

Returns

Return type: real

Area under curve

Example

4.15 *arg*

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

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

4.16 *arg_rad*

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

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

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

4.17 Ascii

Returns the ASCII code for the first letter of the argument

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		

Returns

Return type: real

4.18 asin

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Return the inverse sine of the argument in radians.

4.19 asinh

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Return the inverse hyperbolic sine of the argument.

4.20 AssociateModel

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.

Arguments

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

Returns

Return type: Real

4.21 atan

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

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

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

4.22 atan2

Returns the principal value of the arc tangent of y/x, expressed in radians. To compute the value, the function takes into account the sign of both arguments in order to determine the quadrant. The arguments may be zero.

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Y
2	Real	Yes		X

Argument 1

Y

Argument 2

X

Returns

Return type: Real

atan(y/x) in radians

4.23 atan2_deg

Returns the principal value of the arc tangent of y/x, expressed in degrees. To compute the value, the function takes into account the sign of both arguments in order to determine the quadrant. The arguments may be zero.

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Y
2	Real	Yes		X

Argument 1

Y

Argument 2

X

Returns

Return type: Real

atan(y/x) in degrees

4.24 atan_deg

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

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

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

4.25 atanh

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Return the inverse hyperbolic tangent of the argument.

4.26 avg

Calculates the average of the argument with respect to its reference as defined by:

$$y = \int_0^t \frac{x}{t} dt$$

where x is the argument and t is the reference of x . See [Vector References](#) for details.

The function uses simple trapezoidal integration.

An error will occur if the argument supplied has no reference.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		vector

Returns

Return type: real array

4.27 BoolSelect

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 function [length\(\)](#).

Arguments

Number	Type	Compulsory	Default	Description
1	real	No		Initial check box settings
2	string	No		Labels
3	string	No		Dialog Box Caption
4	string	No		GroupBox Title
5	string	No		Descriptive Text

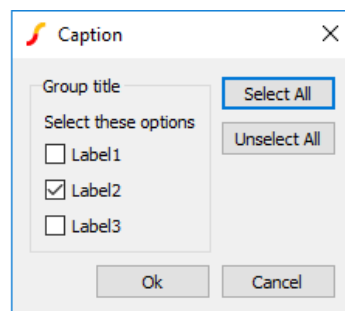
Returns

Return type: real array

Example

The following dialog box is displayed after a call to:

```
BoolSelect([0,1,0], ['Label1', 'Label2', 'Label3'], 'Caption', 'Group title',
+ 'Select these options')
```



See Also

[EditSelect\(\)](#)

[RadioSelect\(\)](#)

[ValueDialog\(\)](#)

4.28 Branch

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.

Arguments

No arguments

Returns

Return type: string

See Also

[NearestInst\(\)](#)

[NetName\(\)](#)

[PinName\(\)](#)

4.29 BuildMclogHTML

Builds an HTML Monte Carlo log file from the XML input file generated by the simulator during a Monte Carlo analysis

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Input file
2	String	Yes		Output file
3	String	Yes	mclog.css	CSS file

Argument 1

XML input file. This is generated by the Monte Carlo analysis. Its path can be retrieved using the [GetAnalysisInfo\(\)](#) function using the outfile parameter.

Argument 2

Path to file to receive HTML generated code

Argument 3

Optional CSS file that may be used to style the HTML. This CSS file will be referenced in the HTML output and not embedded in it. Note that if the CSS file is not found, a default style will be used

Returns

Return type: real

Status: success: 1.0, fail: 0.0

4.30 BuildParameterString

Constructs a string of name=value pairs from two arrays containing the names and values separately.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Parameter names
2	string array	Yes		Parameter values
3	string	No	space and tab	Special characters

Argument 1

Array of parameter names

Argument 2

Array of parameter values

Argument 3

Special characters. Any parameter value containing one or more of the characters specified here will be enclosed in double quotation marks.

Returns

Return type: string

String in form 'name=value name=value ...'

See Also

[ParseParameterString\(\)](#)

[ParseParameterString2\(\)](#)

4.31 BuildSensitivityHTML

Builds a Sensitivity report HTML file from the XML file generated by a sensitivity analysis.

Note that this fil

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Input file
2	String	Yes		Output file
3	Real	No	0	Sort column
4	String	Yes	sensitivity.css	CSS file

Argument 1

XML input file. This is generated by the sensitivity analysis. Its path can be retrieved using the [GetAnalysisInfo\(\)](#) function using the outfile parameter.

Argument 2

Path to file to receive HTML generated code

Argument 3

The resulting HTML file is in the form of a table. By default, the table is sorted in order of case number - column 0. This argument specifies alternative columns to be used for the sort order.

Argument 4

Optional CSS file that may be used to style the HTML. This CSS file will be referenced in the HTML output and not embedded in it. Note that if the CSS file is not found, a default style will be used

Returns

Return type: real

Status: success: 1.0, fail: 0.0

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

4.32 BuildWorstCaseHTML

Builds a Sensitivity report HTML file from the XML file generated by a worst-case analysis.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Input file
2	String	Yes		Output file
3	String	Yes	worstcase.css	CSS file

Argument 1

XML input file. This is generated by the Worst-case analysis. Its path can be retrieved using the [GetAnalysisInfo\(\)](#) function using the outfile parameter.

Argument 2

Path to file to receive HTML generated code

Argument 3

Optional CSS file that may be used to style the HTML. This CSS file will be referenced in the HTML output and not embedded in it. Note that if the CSS file is not found, a default style will be used

Returns

Return type: real

Status: success: 1.0, fail: 0.0

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

4.33 CanOpenFile

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

'copy' mode and 'read' mode differ only in the share access required for the file. 'copy' is less restrictive and will allow copy operations using [CopyFile](#) but will not allow read operations using, for example, [ReadFile\(\)](#).

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		file name
2	string	No	read	options

Returns

Return type: real

4.34 ChangeDir

Change current working directory to that specified by argument.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		New directory

Returns

Return type: real

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

Code	Meaning
0	Success
1	Cannot create directory
2	Invalid disk (windows)

4.35 Char

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Input string
2	real	Yes		Character position

Returns

Return type: string

Example

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

displays result:

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

4.36 CheckLaplaceExpression

Checks a Laplace expression for correctness

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Expression to be analysed

Argument 1

Expression to be analysed

Returns

Return type: real

Integer value from -1 to 4 signifying the validity of the Laplace expression.

Value	Description
-1	Unknown error in expression
0	Expression is valid
1	Syntax error in expression
2	Mismatched parantheses in expression
4	Expression uses an unknown function

4.37 ChooseDir

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.

Arguments

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

Returns

Return type: string

4.38 ChooseDirectory

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 the standard system dialog which includes access to network shares.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Starting Directory

Returns

Return type: string array

4.39 Chr

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)).

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		ASCII code

Returns

Return type: string

4.40 CloseEchoFile

Closes the file associated with the Echo command. For more information, see [OpenEchoFile\(\)](#).

Arguments

No arguments

Returns

Return type:

4.41 CloseFile

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

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		File handle

Argument 1

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

Returns

Return type: real

4.42 CloseSchematic

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

Function returns 1 if successful otherwise returns 0.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Schematic handle

Returns

Return type: real

4.43 CloseSchematicTab

Closes a schematic using its ID. A schematic's ID may be obtained from [OpenSchematic\(\)](#) or [GetSchematicTabs\(\)](#).

Function returns 1 if successful otherwise returns 0.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Schematic ID

Returns

Return type: real

4.44 Coll

Performs the same operation as the `[]`[0] operator if the argument is a multi-division vector. If the argument is a single division vector, this function simply returns the argument unmodified. This function is used in arbitrary fixed probes used for plotting histograms where there is a need to distinguish between single and multi-step Monte Carlo analyses

Arguments

Number	Type	Compulsory	Default	Description
1	Real or complex	Yes		Input vector

Argument 1

Input vector. May be single or multi-division

Returns

Return type: real or complex

Single division vector

4.45 CollateVectors

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.

Arguments

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

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.

Returns

Return type: real or complex array

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.

4.46 CommandStatus

Obtain information about the current script execution context

Arguments

No arguments

Returns

Return type: real array

Four element array. Elements described in the following table:

Index	Description
0	Drag and drop: 1 if current script was called as a result of a drag and drop operation, otherwise 0
1	GUI Action: 1 if the current script was called by a GUI action such as a menu operation. 0, if called by a remote command. Refer to ProcessingGuiAction() for a more detailed explanation
2	Processing accelerator: 1 if the current script was called by an accelerator key, otherwise 0
3	Running startup command: 1 if current script was called by a startup command provide on the SIMetrix.exe command line

See Also

[ProcessingAccelerator\(\)](#)

[ProcessingDragAndDrop\(\)](#)

[ProcessingGuiAction\(\)](#)

4.47 CompareSymbols

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.

Arguments

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

Returns

Return type: real

4.48 ComposeDigital

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.

Arguments

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

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 used to describe how bus vectors are named. The default value 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](#).

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.

Returns

Return type: real vector

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:

BUS1#0 + BUS1#1 * 2 + BUS1#2 * 4 + BUS1#3 * 8.

4.49 ConvertFromBase64

Converts a Base64 expression to string.

Arguments

Number	Type	Compulsory	Default	Description
1	string	true		input

Argument 1

Text input to convert from Base64.

Returns

Return type:

Index	Description
0	

4.50 ConvertHTMLcolourToRGB

Converts a colour definition from HTML format to decimal RGB format.

HTML format is in the form #rrggbb where rr, gg and bb are hexadecimal values describing the red, green and blue content respectively. The decimal RGB value is equal to $bb*65536 + gg*256 + rr$ expressed as a decimal integer.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		HTML Colour

Argument 1

HTML Colour. See description for details

Returns

Return type: String

Decimal RGB colour value. See description for details

4.51 ConvertIsoTimeToUnix

Converts time in form DD/MM/YYYY to Unix Epoch time. This is the number of seconds elapsed since Jan 1, 1970

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Time is ISO format

Argument 1

Time in form YYYY-MM-DD. Will also accept month as 3 character English string (jan, feb, etc)

Returns

Return type: real

Time expressed as the number of seconds elapsed since Jan 1, 1970

4.52 ConvertLocalToUnix

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File path

Returns

Return type: string

This function returns argument 1 but with any back slash characters replaced by forward slash.

See Also

[ConvertUnixToLocal\(\)](#)

4.53 ConvertNumberFromBase64

Converts a Base64 number to string.

Arguments

Number	Type	Compulsory	Default	Description
1	number	true		input

Argument 1

Text input to convert from Base64.

Returns

Return type:

Index	Description
0	

4.54 ConvertNumberToBase64

Converts a number to Base64, which is a text representation of a binary format.

Arguments

Number	Type	Compulsory	Default	Description
1	number	true		input

Argument 1

Number to convert to Base64.

Returns

Return type:

Index	Description
0	

4.55 ConvertRGBcolourToHTML

Converts a colour definition from Decimal RGB format to HTML format.

HTML format is in the form #rrggbb where rr, gg and bb are hexadecimal values describing the red, green and blue content respectively. The decimal RGB value is equal to $bb*65536 + gg*256 + rr$ expressed as a decimal integer.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Decimal RGB Colour

Argument 1

Decimal RGB. See description for details

Returns

Return type: String

HTML colour value. See description for details

4.56 ConvertToBase64

Converts a string to Base64, which is a text representation of a binary format.

Arguments

Number	Type	Compulsory	Default	Description
1	string	true		input

Argument 1

Text input to convert to Base64.

Returns

Return type:

Index	Description
0	

4.57 ConvertUnixTimeToIso

Converts UNIX time to form YYYY-MM-DD. UNIX time the number of seconds elapsed since Jan 1, 1970.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		UNIX time

Argument 1

UNIX time

Returns

Return type: string

Time expressed in form YYYY-MM-DD

4.58 ConvertUnixToLocal

Convert filename to local format using backslash for the directory separator.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File path

Returns

Return type: string

Any forward slash found in the input string is replaced by a back slash.

4.59 CopyDivisionData

›Copies division data from a vector to another vector. Division data are text items attached to vectors that typically define information on multi-step simulations.

Arguments

Number	Type	Compulsory	Default	Description
1	Real or complex array	Yes		Vector to copy division data from
2	Real or complex array	Yes		Vector to receive division data

Argument 1

Source vector for division data. This is expected to be a multi-division vector

Argument 2

Target vector for division data. This is expected to be a multi-division vector

Returns

Return type: Same as argument 2

Vector from argument 2 with division data copied form argument 1

4.60 CopyTree

Copy a directory tree. Requires target to be empty

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Source directory
2	string	Yes		Target directory

Returns

Return type: string

Single string value providing status of operation as follows

Value	Description
success	Operation successful
failed	Operation failed
incomplete	Operation partially completed: some files were not copied
notempty	Target already exists and was not empty
sourcenotexist	Source does not exist
unknown	Unknown error

4.61 CopyURL

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		From URL file
2	string	Yes		To URL file
3	string	No	progress	options

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.

Returns

Return type: string array

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

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.

4.62 *cos*

Returns the cosine of its argument. Result is in radians.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Returns the cosine of its argument. Result is in radians.

4.63 **cos_deg**

Returns the cosine of the argument. Result is in degrees.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Returns the cosine of the argument. Result is in degrees.

4.64 **cosh**

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Return the hyperbolic cosine of the argument specified in radians.

4.65 **CountChars**

Counts characters in a string. Can be used to test for mis-matched parentheses or brackets in an expression

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		String to count
2	string	Yes		Characters to count

Returns

Return type: real array

Each element in the array correspond to the character in the corresponding location in arg 2 and holds the count of that character in the input string.

4.66 CreateDiodeDialog

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	No		Initial values

Argument 1

String array providing initial values for the various controls. The order is 'IF', 'IRM', 'dIf/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.

Returns

Return type: string array

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.

4.67 CreateGraphMeasurement

Creates a graph measurement object. Measurement objects are visible in the measurement window. Typically they display scalar measurements on whole curves.

Arguments

Number	Type	Compulsory	Default	Description
1	String array	Yes		Curve ID and, optionally, graph ID
2	String array	No	All values empty	Values for Label, Expression and Template

Argument 1

First element is the ID of the associated curve. Optional second element specifies a graph ID. If omitted, the measurement object is placed in the currently selected graph.

Argument 2

String array with up to three elements providing values for the Label, Expression and Template properties

Returns

Return type: String

ID of measurement created. Result will be empty if the specified curve does not exist or if the specified graph, if any, does not exist.

See Also

[EditGraphMeasurement\(\)](#)

[FindGraphMeasurement\(\)](#)

4.68 CreateLockFile

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		filename
2	string	Yes		operation

Argument 1

Filename to lock. The lock file created will have the same name with the suffix .lck. The lock file itself will be locked for write and other applications will not be able to delete or write to the file.

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.

Returns

Return type: string

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

4.69 CreateNewTitleBlockDialog

Displays the title block creation dialog.

Arguments

Number	Type	Compulsory	Default	Description
1	string vector	No		Initial display values

Argument 1

Initial display values for the dialog. Each value is in a separate vector element and will start with one of the following prefixes (including the colon ':'):

Prefix	Description
CompanyName:	Company name to appear
Title:	Title of the schematic
Author:	Author of the schematic
Notes:	Notes about the schematic
LayoutStyle:	Either 'Horizontal' or 'Vertical'. Vertical mode will not display an image.
Logo:	Full path to an image to use.
Version:	Schematic version number. Use "<<auto>>" for an automatically assigned version number.
Date:	Schematic version date. Use "<<auto>>" for an automatically assigned version number.

Not all of these values have to be defined. If no values are defined, then the company, author and logo image will attempt to be chosen from option settings.

Returns

Return type: string array

Title block definition. Values are specified one per vector element and have one of the following prefixes (including the colon ':'):

Prefix	Description
CompanyName:	Company name to appear
Title:	Title of the schematic
Author:	Author of the schematic
Notes:	Notes about the schematic
LayoutStyle:	Either 'Horizontal' or 'Vertical'. Vertical mode will not display an image.
Logo:	Full path to an image to use.

Prefix	Description
Version:	Schematic version number. Use “«auto»” for an automatically assigned version number.
Date:	Schematic version date. Use “«auto»” for an automatically assigned version number.

4.70 CreateSharedAxisConnector

Creates a SharedAxis object to connect x-axes. Each grid has its own x-axis but these may be connected to other x-axes in the same graph. Connected x-axes share their properties such as the minimum and maximum limits.

To connect an x-axis, the SharedId property of the axis to the ID returned by this function. X-axes connected to the same ID will be connected.

Arguments

Number	Type	Compulsory	Default	Description
1	String	No	Current graph	Graph ID

Argument 1

Graph ID

Returns

Return type: String

SharedAxis ID

4.71 CreateShortcut

Create a ‘shortcut’ to a file or directory.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path of object
2	string	Yes		Path of link file
3	string	Yes		Description

Argument 1

Path of file or directory which shortcut will point to

Argument 2

Path of shortcut itself.

Argument 3

Description of shortcut

Returns

Return type: string

'Success' or 'Fail'

4.72 CreateTimer

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Command
2	real	Yes		Interval
3	string array	No	2	options

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. String array containing any combination of 'oneshot' and 'echo'. 'oneshot' defines a timer that will trigger only once. 'echo' enables message output in the command shell.

Returns

Return type: real

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

4.73 CurveEditDialog

Open dialog box to edit curve labels and colours.

Arguments

Number	Type	Compulsory	Default	Description
1	String array	Yes		Initial values for dialog box controls

Argument 1

String array consisting of pairs of values. The first element in each pair defines the current label and the second value defines the curve colour using the HTML format. The dialog box will display a single label/colour editor for each pair entered

Returns

Return type: String array

Values entered by user. First array element will contain the text 'restoredefaults' if the Restore Defaults check box is checked. Otherwise this entry will be an empty string.

The remaining entries are in the same format as the argument and define the label/colour choices entered by the user.

4.74 CurveFit

Compares two vectors and returns a normalised difference value. If the vectors are identical, the function returns 0. If one is the exact negative of the other the function returns 1.0. The function is designed to be used for curve fit optimisation. The function is tolerant of vectors with different x-values. If the two vectors do not have coincident end-points, the result will be calculated over the span which overlaps. The function will return empty if there is no overlap.

The function expects both vectors to be X-Y vectors. That is they contain x-values. This is always the case for simulation vectors. For example the vectors generated by a DC sweep simulation have x-values representing the swept parameter. Vectors created by a transient analysis have x-values representing time.

The function will return an empty vector if either vector has no x-values. The [XY\(\)](#) function can be used to create an X-Y vector from simple vectors.

Arguments

Number	Type	Compulsory	Default	Description
1	Real array	Yes		Vector 1 for comparison
2	Real array	Yes		Vector 2 for comparison

Argument 1

Vector 1 for comparison

Argument 2

Vector 1 for comparison

Returns

Return type: Real

Scalar value indicating the difference between the two vectors. 0 means they are identical over their overlapping range. 1.0 means they are equal and opposite

4.75 CurveFitLog

As [CurveFit\(\)](#) but applies a logarithm function to both X and Y data prior to evaluating the curve fit function

Arguments

Number	Type	Compulsory	Default	Description
1	Real array	Yes		Vector 1 for comparison
2	Real array	Yes		Vector 2 for comparison

Argument 1

Vector 1 for comparison

Argument 2

Vector 1 for comparison

Returns

Return type: Real

Scalar value indicating the difference between the two vectors. 0 means they are identical over their overlapping range. The function will return 1.0 if one argument is the reciprocal to the other

4.76 CurveFitLogX

As [CurveFit\(\)](#) but applies a logarithm function to the X-data prior to evaluating the curve fit function

Arguments

Number	Type	Compulsory	Default	Description
1	Real array	Yes		Vector 1 for comparison
2	Real array	Yes		Vector 2 for comparison

Argument 1

Vector 1 for comparison

Argument 2

Vector 1 for comparison

Returns

Return type: Real

Scalar value indicating the difference between the two vectors. 0 means they are identical over their overlapping range. The function will return 1.0 if one argument is equal and opposite to other

4.77 CurveFitLogY

As [CurveFit\(\)](#) but applies a logarithm function to the Y-data prior to evaluating the curve fit function

Arguments

Number	Type	Compulsory	Default	Description
1	Real array	Yes		Vector 1 for comparison
2	Real array	Yes		Vector 2 for comparison

Argument 1

Vector 1 for comparison

Argument 2

Vector 1 for comparison

Returns

Return type: Real

Scalar value indicating the difference between the two vectors. 0 means they are identical over their overlapping range. The function will return 1.0 if one argument is the reciprocal of the other

4.78 cv

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](#). 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.

This function is identical to [GetCurveVector\(\)](#) and is convenient in situations where a short expression is desirable.

Arguments

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

Returns

Return type: real array

4.79 CyclePeriod

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

Arguments

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

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.

Returns

Return type: real

4.80 Date

Returns the current date in the format specified.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	'locale'	format

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.

Returns

Return type: string

4.81 db

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Returns $20 \times \log_{10}(\text{mag}(x))$

4.82 DCSourceDialog

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

Return value is the user’s entry

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Initial value

Returns

Return type: real

4.83 DCSourceDialogStr

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

Return value is the user’s entry

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Initial value

Returns

Return type: string

4.84 DefineADCDialog

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)

Element index	Description
3	Offset voltage (default 0)
4	Range (default 5)

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		initial values

Returns

Return type: real array

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.

4.85 DefineArbSourceDialog

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)

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		initial values

Returns

Return type: string array

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

4.86 DefineBusPlotDialog

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

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Initial values
2	string	No		options

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: '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	
11	Axis type: '0' Auto select '1' Use separate Y-axis '2' Use separate grid '3' Digital	
12	Axis name	

Index	Description	Default
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.

Returns

Return type: string array

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.

4.87 DefineCounterDialog

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)

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		initial values

Returns

Return type: real array

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.

4.88 DefineCurveDialog

Opens the dialog box used to define a curve for plotting. See menu **Probe | Add Curve...** or **Plot | Add Curve...** in the graph window.

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\(\)](#) and [EditProbeDialog\(\)](#). 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	
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 '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

Index	Purpose	Notes	Default
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	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.	
25	Detached x-axis	State of Detached x-axis check box	<<empty>>

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

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		initial values

Returns

Return type: string array

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

4.89 DefineDACDialog

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Initial values

Returns

Return type: real array

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.

4.90 DefineFourierDialog

Opens the Define Fourier dialog box used to specify a fourier transform. This is similar to the [Define Curve dialog\(\)](#) but has an extra tabbed sheet to define the fourier analysis options. Select menu **Probe | Fourier | Arbitrary...** 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. The remaining are described in the following table:

Index	Description	Default
0-24	See DefineCurveDialog()	
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
34	Use specified span: TRUE/FALSE	FALSE
35	Know fundamental frequency: TRUE/FALSE	FALSE
36	Resolution	100

Index	Description	Default
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.

Arguments

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

Returns

Return type: string array

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

4.91 DefineFourierProbeDialog

Opens dialog to edit fixed Fourier Probe

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Values to initialise dialog

Argument 1

Values to initialise dialog

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.

Index	Purpose	Notes	Default
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	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 Define Curve sheet	<<empty>>
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	Disable curve name	If true, the curve label box is disabled	'false'
9	Display order	String to specify curve display order	<<empty>>
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 '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'

Index	Purpose	Notes	Default
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>>
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.	
25	Curve colour	Colour of curve as an RGB value. May be passed directly to the .GRAPH colour parameter	<<empty>>
26	All analyses disabled	Disables all analyses for this probe.	0
27	Help context id		
28	Measurement type	Not used with this function	
29	Output type	Not used with this function	
30	Edge type	Not used with this function	
31	Probe type	Not used with this function	
32	AC power mode	Not used with this function	
33	createMode	Not used with this function	
34	probeExpression	Not used with this function	
35	Fundamental frequency	Frequency setting in Signal info group	

Index	Purpose	Notes	Default
36	Start frequency	Start frequency entry in Frequency display group	
37	Stop frequency	Stop frequency entry in Frequency display group	
38	Number of FFT points	Num. points entry in FFT interpolation group	
39	FFT order	Order entry in FFT interpolation group	
40	Fourier method	Possible values: 'continuous' Use continuous fourier 'interpolated' Use interpolated FFT	'continuous'
41	Window function	Possible values: 'rectangular' 'hanning' 'hamming' 'blackman'	'hanning'
42	Start of data span	Start value in Data span group	0
43	End of data span	End value in Data span group	0.01
44	Use specified span	True/False : False Use transient analysis parameters True Specify	
45	Know fundamental frequency	Setting in Signal info group	False
46	Resolution	Resolution entry in Frequency display group	100
47	Plot options	'mag', 'db' or 'phase'	'mag'
48	Run time calculation time limit		
49	Default resolution		
50	Number of divisions		

Returns

Return type: string array

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

4.92 DefineIdealTxDialog

Opens a dialog box to define an ideal transformer.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Inductance factor, coupling factors, number of windings
2	real array	Yes		Number of turns for primary windings
3	real array	Yes		Number of turns for secondary windings
4	string	No		options

Argument 1

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

- 0 Inductance factor, AL. $L = AL * N*N$
- 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 values representing the number of turns for each primary winding.

Argument 3

Real array of values representing the number of turns for each secondary winding.

Argument 4

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

Returns

Return type: real array

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.

Notes

The dialog box design has changed significantly with version 8.20 compared to earlier versions. Functionally it is compatible if the number of turns in primary 1 is kept at 1. The earlier designs returned turns ratios and no facility was provided to set the number of turns for the first primary.

4.93 DefineLaplaceDialog

Opens a dialog box to define a Laplace transfer function. This is used to interface to the simulator devices `s_xfer` and `Laplace`. The former implements a Laplace transfer function using a network of integrators while the latter uses frequency-time domain conversion by convolution. This dialog box provides a unified user interface to both devices.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Initial settings
2	string array	No		Layout options

Argument 1

The argument is a string array of length 14 that defines the initial settings. The meaning of each element is as follows:

Index	Description
0	Laplace expression. (contents of “Define output using s variable” box)
1	Device type: <ul style="list-style-type: none"> 0 Transfer function 1 Impedance - V/I 2 Admittance - I/V
2	Input type: <ul style="list-style-type: none"> 0 Single ended voltage 1 Single ended current 2 Differential voltage 3 Differential current
3	Output type: <ul style="list-style-type: none"> 0 Single ended voltage 1 Single ended current 2 Differential voltage 3 Differential current
4	Frequency scale factor
5	Method: 0 Lumped network, 1 Convolution
6	Convolution size: Size of convolution as power of 2. Value from 9 to 30. Dialog will limit maximum value according to available memory
7	Enable error control: 0 Error control disabled, 1 Error control enabled
8	Relative tolerance
9	Absolute tolerance

Index	Description
10	Impulse extraction method: 0 Try all methods 1 Analytic 2 Inverse FFT 3 Stehfest
11	Inverse FFT size as power of 2. . Value from 9 to 32. Dialog will limit maximum value according to available memory
12	Additional Delay
13	Enable diagnostics

Argument 2

Options for setting layout. Set to 'allowconvolution' to enable the advanced features relating to convolution method. If this option is not provided, a simplified version of the dialog will be displayed that allows only lumped network definitions to be entered

Returns

Return type: string array

The function returns a string array of length 14 with the same format as the argument described above. If the user selects "Cancel" the function returns an empty vector.

4.94 DefineLogicGateDialog

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

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		initial settings

Returns

Return type: real array

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.

4.95 DefinePerfAnalysisDialog

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		initial values

Returns

Return type: string array

4.96 DefineRegisterDialog

Opens a dialog box to define a bus register.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		initial settings

Argument 1

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

Returns

Return type: real array

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.

4.97 DefineRipperDialog

Opens a dialog box to define a schematic bus ripper.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		initial settings
2	string array	Yes		list of style box items

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

Returns

Return type: string array

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.

4.98 DefineSaturableTxDialog

Opens a dialog box to define a saturable transformer.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Core material info
2	string array	Yes		Part number info
3	real array	Yes		Winding ratios
4	real array	Yes		Initial values

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

Returns

Return type: real array

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.

4.99 DefineShiftRegDialog

Open a dialog box to define a shift register.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		initial settings

Argument 1

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

Returns

Return type: real array

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.

4.100 DefineSimplisMultiStepDialog

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

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Configuration
2	string array	Yes		Sweep values

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'

Index	Description
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.

Returns

Return type: string array

4.101 DeleteConfigCollection

Deletes an entire section in the configuration file.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		section name

Argument 1

Name of section to be deleted.

Returns

Return type: real

Returns the number of entries successfully deleted.

4.102 DeleteTimer

Deletes a timer

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Timer ID

Argument 1

Timer ID as returned by [CreateTimer\(\)](#)

Returns

Return type: real

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

4.103 DeleteTouchstone

Deletes data loaded by [LoadTouchstone\(\)](#)

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		ID of Touchstone data

Argument 1

ID of Touchstone data as returned by [LoadTouchstone\(\)](#). If 0, will delete all available Touchstone data

Returns

Return type: Real

Number of data items deleted.

See Also

[ReadTouchstone\(\)](#)

[GetTouchstoneErrors\(\)](#)

[LoadTouchstone\(\)](#)

4.104 DeleteTree

Delete an entire directory tree

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		directory to delete

Returns

Return type:

Single string value providing status of operation as follows

Value	Description
success	Operation successful
failed	Operation failed
incomplete	Operation partially completed: some files were not deleted
sourcenotexist	Source does not exist
unknown	Unknown error

Example

See Also

[DeleteTreeProgress\(\)](#)

4.105 DeleteTreeProgress

Delete an entire directory tree, while showing a progress box

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		directory to delete

Returns

Return type:

Single string value providing status of operation as follows

Value	Description
success	Operation successful
failed	Operation failed
incomplete	Operation partially completed: some files were not deleted
sourcenotexist	Source does not exist
aborted	Operation aborted by user
unknown	Unknown error

Example

See Also

[DeleteTree\(\)](#)

4.106 DelSchemProp

Deletes a user-defined schematic window property created using WriteSchemProp.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name
2	number	No		Handle

Argument 2

Handle to a schematic.

Returns

Return type: real

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
- 2 Property is not deleteable

4.107 DescendDirectories

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Directory

Returns

Return type: string array

4.108 DescendHierarchy

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	'Ref'	Property used to report 'where used' references
2	real	No	-1	Schematic ID
3	string			options

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 ID as returned by the [OpenSchematic\(\)](#) 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.

Argument 3

If set to 'pathtypes' will return information on the type of path. Possible values are 'absolute', 'relative' and 'symbolic'

Returns

Return type: string array

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. 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.

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.

4.109 DialogDesigner

Simple dialog designer that generates an XML dialog definition. The dialog shows the dialog as a tree, where the user can drag and drop items in the tree, add groups and add tabs. A preview of the dialog is shown alongside.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	<<empty string>>	Initial XML definition

Argument 1

This optional argument can contain a basic XML definition of the dialog. Note that XML nesting is not processed and all elements are added to the root of the tree.

Returns

Return type: string

An XML file describing the dialog.

4.110 *diff*

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](#).

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		vector

Returns

Return type: real array

4.111 *DirectoryIsWriteable*

Tests whether or not a directory can be written to by creating a temporary writeable file in that directory. If the file is successfully created the directory is deemed to be writeable

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Directory to test

Returns

Return type: real

1 if directory is writeable otherwise 0

4.112 Distribution

Returns a random number with a distribution defined by a lookup table. This function is intended to be used for SIMPLIS Monte Carlo analyses and would typically be used in device value expressions.

This function is only available in the Simulator process and cannot be called from scripts running in the context of the front end. The function is only active when used by the netlist pre-processor with Monte Carlo analysis enabled. When used in other contexts, the function returns 1.0.

A similar function is available for SIMetrix Monte Carlo analyses, but the syntax is slightly different. Refer to the Simulator Reference Manual for further details.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Tolerance
2	real array	Yes		Distribution definition

Argument 1

Tolerance - in effect scales the extent of the distribution defined in argument 2.

Argument 2

Lookup table organised in pairs of values.

The first value in the pair is the deviation. This should be in the range +1 to -1 and maps to the output range. So +1 corresponds to an output value of +tolerance and -1 corresponds to -tolerance. Each deviation value must be greater than or equal to the previous value. Values outside the range +/- 1 are allowed but will result in the function being able to return values outside the tolerance range.

The second value in the pair is the relative probability and must 0 or greater.

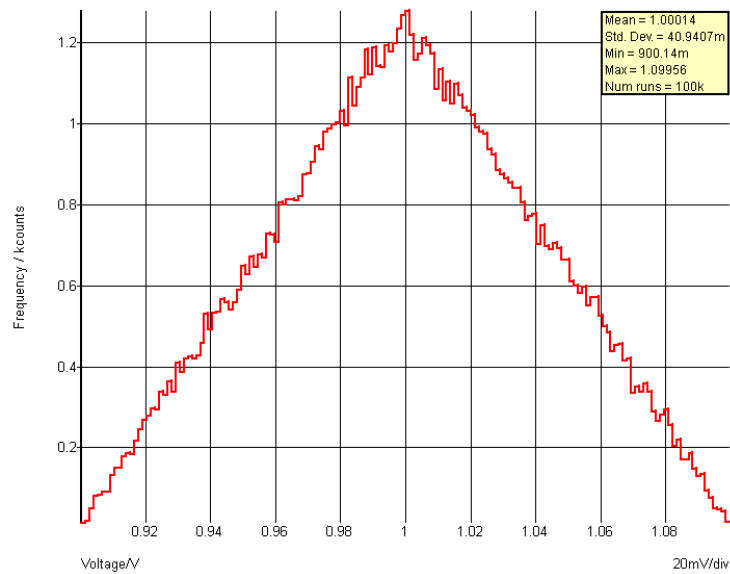
There is no limit to the number of entries in the table

Returns

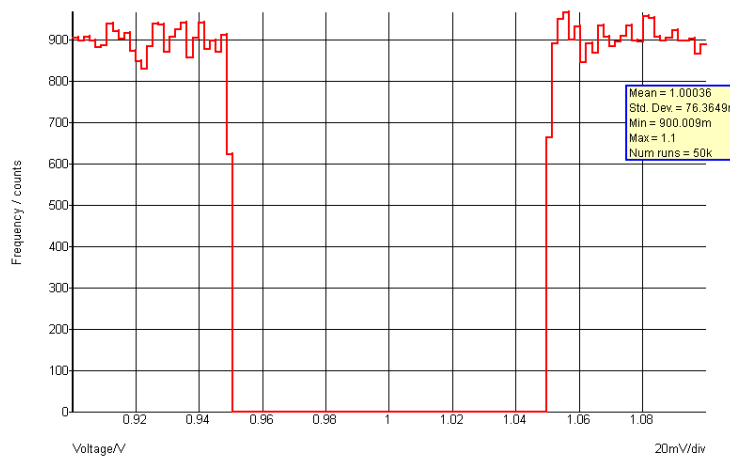
Return type: real

Example

distribution(1.0, [-1,0, 0,1, 1,0]) - see graph below:



distribution(1.0, [-1,1, -0.5,1, -0.5,0, 0.5,0, 0.5,1, 1,1])



Notes

If multiple instances of a particular distribution are needed, a variable of the lookup table may be defined. For example:

```
.VAR binomial = {[-1,1, -0.5,1, -0.5,0, 0.5,0, 0.5,1, 1,1]}
```

The above can be placed in the F11 window of a SIMPLIS schematic. Then to access a binomial distribution for a component value, use something like:

```
{ 1k * distribution(0.1, binomial) }
```


The above defines a value of 1k with a 10% tolerance using the binomial distribution defined by the lookup table 'binomial'.

The function `UD()` is an alias to this function and may be more convenient.

See Also

`Gauss()`

`GaussTrunc()`

`Unif()`

`UD()`

`WC()`

4.113 EditArcDialog

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

Arguments

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

Argument 1

Initial value for start to finish angle.

Argument 2

Initial value for ellipse height/width.

Returns

Return type: real array

If the user selects Cancel the function returns an empty vector, otherwise the following real array of length 2 is produced:

Index	Description
0	Start to finish angle
1	Ellipse height/width

4.114 EditAxisDialog

Opens a dialog box used to edit graph axes

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		initial settings

Argument 1

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\(\)](#) and [EditProbeDialog\(\)](#) 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
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	

Returns

Return type: string array

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

4.115 EditBodePlotProbeDialog

UI function for editing Bode plot fixed probes.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	No		Initialisation

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

Returns

Return type: string array

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

4.116 EditBodePlotProbeDialog2

Opens dialog box for editing the advanced Bode plot probe.

Arguments

Number	Type	Compulsory	Default	Description
1	String array	Yes		Initial values for dialog box controls

Argument 1

String array with 27 values as defined below:

- 0 Graph name
- 1 History depth
- 2 Gain enabled
- 3 Gain curve label
- 4 Gain y-axis label
- 5 Gain vertical scale 0:db, 1:Linear
- 6 Gain auto y-axis limits
- 7 Gain auto grid spacing
- 8 Gain vertical axis maximum limit
- 9 Gain vertical axis minimum limit
- 10 Gain vertical axis grid spacing
- 11 Gain curve colour
- 12 Phase enabled
- 13 Phase curve label
- 14 Phase y-axis label
- 15 Phase polarity 0:normal 1: -180 degrees
- 16 Phase auto axis limits
- 17 Phase auto grid spacing
- 18 Phase vertical axis maximum limit
- 19 Phase vertical axis minimum limit
- 20 Phase vertical axis grid spacing
- 21 Phase curve colour
- 22 Num grids 0:1 grid 1:2 grids
- 23 Vertical order 0:phase above gain, 1: gain above phase
- 24 Save settings
- 25 Set tab/caption to graph name
- 26 Separate curves

Returns

Return type:

String array in same format as argument 1 providing user's selected values. Returns an empty vector if the user cancels.

4.117 EditCrosshairDimensionDialog

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

The Properties sheet behaves in the same way as the [EditObjectPropertiesDialog\(\)](#) function 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
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 P2P1 AutoAutomatic, 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Property names
2	string array	Yes		Property values
3	string array	No		Property types

Returns

Return type: string array

4.118 EditCurveMarkerDialog

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

The Properties sheet behaves in the same way as the [EditObjectPropertiesDialog\(\)](#) function 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

Property Name	Affects Control
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.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Property names
2	string array	Yes		
3	string array	No		Property types

Argument 2

Property values

Returns

Return type: string array

4.119 EditDeviceDialog

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

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		options/initial settings
2	string array	Yes		devices
3	string array	No	<<empty>>	parameter names
4	string array	No	<<empty>>	parameter values

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.

Returns

Return type: string array

If the user selects Cancel the function returns an empty vector, otherwise returns a string array.

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.

4.120 EditDignitDialog

Opens a dialog box used to define a digital initial condition

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		initial setting

Argument 1

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

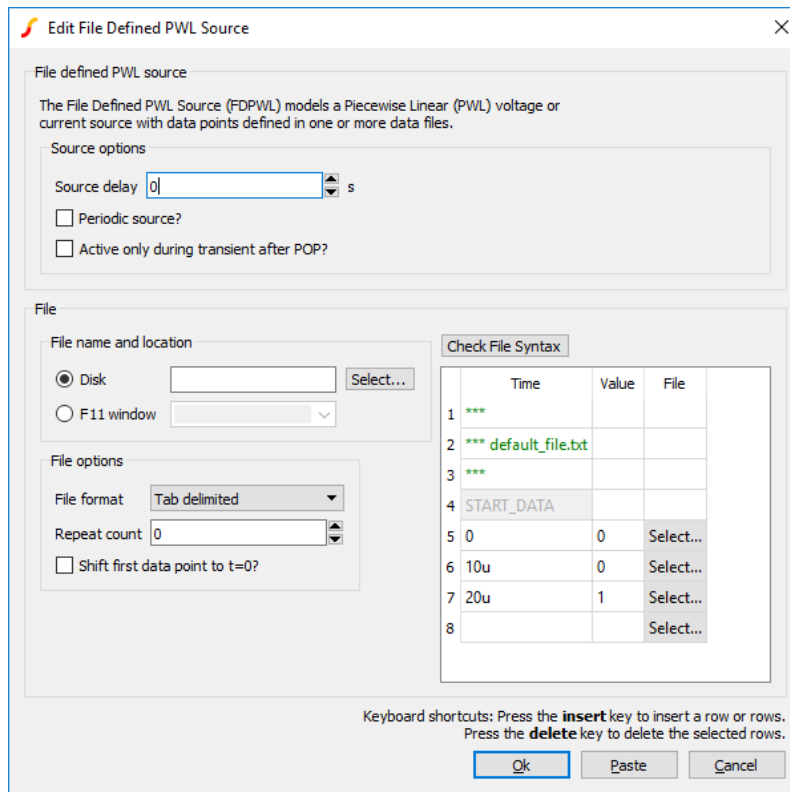
Returns

Return type: real array

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.

4.121 EditFileDefinedPWLDIALOG

Opens the dialog box shown below allowing the entry of X-Y pairs intended for the definition of file defined piece-wise linear sources.



Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		file content
2	string	No		Source options
3	string	No		Options

Argument 1

File content used to initialise table represented as a string array. The above example would be displayed after a call to:

```
Show EditFileDefinedPWLDIALOG(['***', '*** default_file.txt', '***', 'START_DATA', '0 0', '10u 0', '20u 1'], ['0', '0', '0', ' ', 'Disk'])
```

Argument 2

Up to seven element string array to define source parameters:

Index	Description
0	Source delay parameter
1	Checkbox state for Periodic source checkbox.

Index	Description
2	Checkbox state for Active only during transient after POP? checkbox.
3	Filename.
4	File location used to initialize the radio selection. 'Disk' or 'F11'.
5	Reference designator for souce, used in error messages.
6	Pipe () delimited list of F11 filenames used to populate the F11 window combobox.

Argument 3

Up to six element string array to define box labels:

Index	Description
0	Box caption. Default: 'Edit File Defined PWL Source'
1	Label for X-Values column. Default: 'Time'
2	Label for Y-Values column. Default: 'Value'
3	Help context id. Default: '-1' (no help button shown)
4	Minimum number of segments. Default = '2'
5	Maximum number of segments. Default = '100001'

Returns

Return type: string array

The function returns the edited file as a string array. The first element of the return contains the source parameters, concatenated in a semi-colon delimited list of 5 elements:

Index	Description
0	Source delay parameter
1	Checkbox state for Periodic source checkbox.
2	Checkbox state for Active only during transient after POP? checkbox.
3	Filename.
4	File location radio button state. 'Disk' or 'F11'.

The file contents start at the second element (index=1). You can slice the file contents out of the return array with the [Range\(\)](#) function:

```
Let fileContents = Range( return , 1 )
```

String arrays can be written to file with the [Show](#) command.

```
Show /plain /file points.txt fileContents
```

4.122 EditFreeTextDialog

This function is almost identical to the [EditCurveMarkerDialog\(\)](#) functions except for some changes to the aesthetics of the dialog box.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Property names
2	string array	Yes		Property values
3	string array	No		Property types

Returns

Return type: string array

4.123 EditGraphMeasurement

Edits the Label, Expression and Template properties of a measurement object.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Measurement object ID
2	String array	Yes		New values for Label, Expression and Template properties

Argument 1

Measurement object ID

Argument 2

String array with up to three values providing new values for Label, Expression and Template properties respectively

Returns

Return type: Real

1 if successful. 0 if provided ID is invalid

See Also

[CreateGraphMeasurement\(\)](#)

[FindGraphMeasurement\(\)](#)

4.124 EditGraphTextBoxDialog

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\(\)](#) 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Property names
2	string array	Yes		Property values
3	string array	No		Property types

Returns

Return type: string array

4.125 EditJumperDialog

Opens edit jumper dialog

Arguments

Number	Type	Compulsory	Default	Description
1	String array	Yes		Initial values

Argument 1

Three element array providing initial values as defined below:

- 0 Number of positions. 0: 1 position, 1: 2 positions
- 1 Number of poles
- 2 Jumper position. 0: Open, 1: Position 1, 2: Position 2

Returns

Return type: String array

String array in same format as argument providing user selections

4.126 EditLegendBoxDialog

Opens a dialog intended for editing the characteristics of a graph legend.

The Properties sheet behaves in the same way as the [EditObjectPropertiesDialog\(\)](#) 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Property names
2	string array	Yes		Property values
3	string array	No		Property types

Returns

Return type: string array

4.127 EditObjectPropertiesDialog

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

Number	Type	Compulsory	Default	Description
1	string array	Yes		Property names
2	string array	Yes		Property values
3	string array	No		Property types
4	string array	No		Options
5	string	No	<<empty>>	Override Style
6	string	No	<<empty>>	Available override styles

Argument 1

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 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.
item1item2item3l...	Up to six items separated by the 'l' symbol. When the user double clicks a property so defined, a dialog showing a number of radio buttons is displayed labelled item1, item2 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'

Index	Description	Default
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.

Argument 5

If set, this specifies the style the property should use when being displayed on the schematic.

Argument 6

A set of styles that can be chosen between if setting an override style for a property. These styles are chosen from those styles in the Style Library that have the override style flag checked.

Returns

Return type: string array

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

4.128 EditPinDialog

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		initial pin name
2	string	No	'256'	initial flags value
3	string	No		not used
4	string	No		not used
5	string	No	<<empty>>	font override style
6	string	No	<<empty>>	available font override styles

Argument 1

Specifies the initial value for the Pin name entry

Argument 2

Specifies the initial value for the remaining controls using the property attributes flag. See [Attribute Flags in the Prop command](#) for details.

Returns

Return type: string

If the user selects Cancel the function returns an empty vector, otherwise The function returns a string array of length 2.

Index	Description
0	Flags value defining justification and property attributes
1	Pin name

4.129 EditPotDialog

Opens a dialog to define a potentiometer

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		initial settings

Argument 1

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

Returns

Return type: real array

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

4.130 EditProbeDialog

Opens a dialog to define a schematic fixed probe

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		initial settings

Argument 1

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\(\)](#) and [DefineCurveDialog\(\)](#). 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	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>>
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 LogLin/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 LogLin/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’

Index	Purpose	Notes	Default
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>>
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>>
26	All analyses disabled	Disables all analyses for this probe.	0
27	Physical Probe Type		
28	Measurement type		
29	Output type		
30	Edge type		
31	Probe type		
32	AC power mode		
33	Create mode		
34	Probe expression		
35	Multi-step mode		
36	Number of histogram bins		
37	Use default histogram bins		
38	Display advanced statistics		
39	Set graph tab/caption to name		
40	Separate curves		
41	Persistence default		

Returns

Return type: string array

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

4.131 EditPropertyDialog

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes	<<empty>>	property name
2	string	No	0	initial property flags
3	string	No	<<empty>>	initial property value
4	string	No		option
5	string	No	<<empty>>	font override style
6	string	No	<<empty>>	available font override styles

Argument 1

Specifies the property name and this is displayed at the top left of the box. This cannot be edited by the user.

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 in the Prop command](#).

Argument 3

Argument initialises the Value box

Returns

Return type: string array

String array of length 2 providing the users settings, or empty vector if Cancel is pressed.

Index	Description
0	Flags value
1	Property value

4.132 EditReactiveDialog

Opens a dialog box designed to edit inductors and capacitors.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Initial values
2	string	Yes		Options
3	string	No		Parameter names
4	string	No		Parameter values

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
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.

Returns

Return type: string array

The function returns a string array in the following form:

Index	Description
0	Value in Result box

Index	Description
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')

4.133 EditSelect

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.

Arguments

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

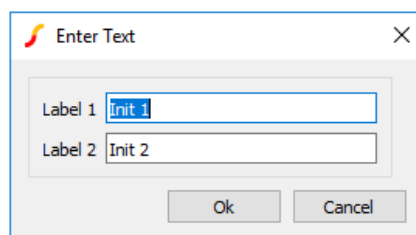
Returns

Return type: string array

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\(\)](#)

[RadioSelect\(\)](#)

[ValueDialog\(\)](#)

4.134 EditSimplisLaplaceFilterDialog

Opens dialog box fro editing SIMPLIS Laplace filters

Arguments

Number	Type	Compulsory	Default	Description
1	String array	Yes		Initial values

Argument 1

25 element array to define initial values

Returns

Return type: String array

25 element array in same format as argument 1 providing user's selections

4.135 EditSimplisMosfetDriverDialog

Opens a specialized dialog used to edit the parameters for a SIMPLIS multi-Level MOSFET Driver. See internal script *simplis_edit_mosfet_driver* for an application example of this function.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	No		Initial values
2	string array	No		Caption, combo options, Help context id

Argument 1

String array providing initial values for the various controls. The order is:

Index	Purpose	Notes	Default
0	LEVEL	The model level	1

Index	Purpose	Notes	Default
1	INVERTING	Inverting flag	0
2	USE_DELAY	Delay flag	1
3	THRESHOLD	The input threshold	2.5
4	HYSTWD	The input hysteresis	1.0
5	RISE_DELAY	The rising edge delay	15n
6	FALL_DELAY	The falling edge delay	10n
7	HS_RDSON	The upper switch RDS(on) for Level 0 and 1 models	1
8	HS_RSAT	The upper switch saturation resistance for Level 0 and 1 models	10Meg
9	HS_ISAT	The upper switch saturation current for Level 0 and 1 models	2
10	LS_RDSON	The lower switch RDS(on) for Level 0 and 1 models	1
11	LS_RSAT	The lower switch saturation resistance for Level 0 and 1 models	10Meg
12	LS_ISAT	The lower switch saturation current for Level 0 and 1 models	3
13	IC	The initial condition of the upper switch.	<<empty>>
14	HS_ROFF	The upper switch off resistance for Level 0 and 1 models	10Meg
15	LS_ROFF	The lower switch off resistance for Level 0 and 1 models	10Meg
16	HS_VON	The upper switch on-state voltage	0
17	LS_VON	The lower switch on-state voltage	0
18	HS_RDSON_L2	The upper switch RDS(on) for Level 2 models	10
19	HS_R2_L2	The upper switch resistance for the second PWL segment	500m
20	HS_RSAT_L2	The upper switch saturation current for Level 2 models	10Meg
21	HS_V1_L2	The voltage where the upper switch transitions from the 1st to 2nd PWL segments	500m
22	HS_ISAT_L2	The upper switch saturation current for Level 2 models	1
23	LS_RDSON_L2	The lower switch RDS(on) for Level 2 models	10
24	LS_R2_L2	The lower switch resistance for the second PWL segment	500m
25	LS_RSAT_L2	The lower switch saturation current for Level 2 models	10Meg
26	LS_V1_L2	The voltage where the lower switch transitions from the 1st to 2nd PWL segments	100m
27	LS_ISAT_L2	The lower switch saturation current for Level 2 models	3

Argument 2

Index	Purpose	Notes	Default
0	Caption	The dialog box caption	“Edit Multi-Level MOSFET Driver”
1	Combo options	Combo items for the initial conditions box.	<<empty>>
2	Help context id	The help context id, used for the built-in Multi-Level MOSFET Driver.	<<empty>>

Returns

Return type: string array

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.

4.136 EditStylesDialog

Opens the Edit Styles dialog. This is a system function and is unsupported.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Style names
2	string array	Yes		Style info
3	string array	Yes		Line types available
4	string array	No	empty string	Hidden default styles for viewer
5	string array	No	empty string	Flags for hiding buttons
6	string array	No	empty string	Global style info
7	string	No	empty string	Editor settings

Argument 2

Style information for each style name specified in argument 1. Each element in the array is matched to the corresponding element in argument 1 and must be in the form:

Name|LineType|LineThickness|LineColour

Argument 3

Each array element is a different line type available to all styles. Options are: Solid, Dash, Dot, DashDot, DashDotDot.

Argument 4

Default styles to use in the preview window that are not shown or editable in the dialog. Only required to ensure the correct default wire, symbol and annotation styles are applied.

Each element in the array is a full style definition, in the form:

```
StyleName|lineColour:[lineColour] lineType:[lineType]
lineThickness:[lineThickness] fontFamily:[fontFamily]
fontItalics:[fontItalics] fontBold:[fontBold] fontColour:[fontColour]
fontSize:[fontSize] propertyStyle:[propertyStyle]
fontOverline:[fontOverline] fontUnderline:[fontUnderline]
```

StyleName values can be either: DefaultWire, DefaultInstance, DefaultAnnotation.

Argument 5

Optional flags for hiding buttons in the dialog. The flags are:

Flag	Behaviour
noadd	Hides the <i>New...</i> button.
noduplicate	Hides the <i>Duplicate</i> button.
noedit	Hides the <i>Edit Name...</i> button.

Argument 6

Global style information, used for reverting local styles back to their global settings. Each row is a separate style, defined in the same form as argument 4. Any style name is allowed.

Argument 7

If set to “FontOnly”, only font settings will be displayed within the editor.

Returns

Return type: string array

String vector of updated styles if successful, or an empty string if cancel is selected.

Each element in the array is a different style. Styles are in the form:

```
StyleName|lineColour:[lineColour] lineType:[lineType]
lineThickness:[lineThickness] fontFamily:[fontFamily]
fontItalics:[fontItalics] fontBold:[fontBold] fontColour:[fontColour]
fontSize:[fontSize] propertyStyle:[propertyStyle]
fontOverline:[fontOverline] fontUnderline:[fontUnderline]
```

4.137 EditSymbolBusDialog

Arguments

Number	Type	Compulsory	Default	Description
1	Real array	Yes		Initial values for start and size of bus
2	String	No		Options

Argument 1

Two element array providing initial values for start and size of bus

Argument 2

Set to 'fixedSize' to disable size box. The box will be initialised but the user will not be able to edit it

Returns

Return type: Real array

Two element array providing user selected start and size values

4.138 EditTimer

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

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Timer ID
2	string	Yes		action
3	real	No		Value

Argument 1

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

Returns

Return type: real

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.

4.139 EditWaveformDialog

Opens a dialog designed for editing a time domain waveform. This function has been superseded by [EditWaveformStrDialog\(\)](#) but is retained to support old designs.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Time/frequency initial values
2	real	No		Vertical initial values
3	string array			options

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: <ul style="list-style-type: none"> 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

Index	Description
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

Argument 3

String array up to length 2 which may specify either of these options

Simulator mode - either 'SIMetrix' or 'SIMPLIS'

Initial pulse mode - set to 'initialpulse'

Returns

Return type: string array

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

4.140 EditWaveformStrDialog

Opens a dialog box 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Time/frequency initial values
2	string array	No		Vertical initial values
3	string array	No		Options

Argument 1

Initial values for the controls in the Time/Frequency group box. Values must be entered as strings and may be in the form of expressions enclosed with curly braces as well as literal constants. Up to 10 elements defined as follows:

Index	Description
0	Integer from 0 to 8, specifies wave shape as follows: <ul style="list-style-type: none"> 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. Values must be entered as strings and may be in the form of expressions enclosed with curly braces as well as literal constants. Up to 5 elements defined as follows:

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

Argument 3

String array which may contain any combination of:

Name	Description
simplis	Select SIMPLIS mode. This shows the "Source idle during POP and AC analyses" check box
initialpulse	If true, means read the values for initial and pulse and use these to derive values for offset and amplitude. If false, read the values for offset and amplitude and use these to derive values for initial and pulse. This is set unconditionally if arg1 is missing or has size <5. Otherwise it is set by a flag in arg2 if present otherwise it is false.

Returns

Return type: string array

String array with 16 elements. Elements 0 - 9 as for argument 1, elements 10-14 as for argument 2. Element returns the state of the "Source idle during POP and AC analyses" check box.

4.141 ElementProps

Returns an array of strings holding the names of all properties of an instance. The functions [PropValue\(\)](#) or [PropValues2\(\)](#) can be used to find values of these properties.

This is a generalisation of [InstProps\(\)](#), in that it will return the properties for any selected schematic element.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Property name
2	string	No		Property value
3	real	No	-1	Schematic ID

Argument 1

Property name to identify element. Along with parameter 2, if these arguments are not provided, the selected element, if any, will be used instead. If there are no selected elements or no elements that match the arguments, the function will return an empty vector. If the arguments identify more than one element, 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 2

Property value to identify element. Along with parameter 1, if these arguments are not provided, the selected element, if any, will be used instead. If there are no selected elements or no elements that match the arguments, the function will return an empty vector. If the arguments identify more than one element, 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 ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string array

Array of strings with property values. Returns empty value if no match to property name and value is found. Also returns empty value if the schematic ID is invalid.

4.142 EncodeImageToBase64

Returns the Base64 binary encoding of a png or bmp image file as a text string. The first argument can be a full file path name to the image, a directory path, or left blank. The latter two options will prompt a user with a dialog to choose the desired png or bmp file.

Arguments

Number	Type	Compulsory	Default	Description
1	String	No		File path name of png or bmp file

Argument 1

Full file path name, file directory path, or empty.

Returns

Return type: String

String with base64 encoding of png or bmp image if conversion is successful. In case of a failure to convert the file, returns a integer error code.

Integer Error Code	Description
0	File doesn't exist
1	Incorrect file format - supported formats are PNG and BMP
2	Failed to convert file

You can test the return value with the `IsStr()` function to catch and handle errors.

4.143 EnterTextDialog

Opens a dialog box allowing the user to enter lines of text.

Arguments

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

Argument 1

The argument specifies the initial text and the dialog box's caption as follows:

- 0 Initial text
- 1 Dialog box caption

Returns

Return type: string

The function returns the text entered by the user.

4.144 EpochTime

Returns the number of seconds elapsed since midnight, January 1, 1970.

Arguments

No arguments

Returns

Return type: real

Example

Notes

The return value has a numerical resolution of 1 ms but the useable resolution is system dependent and usually much coarser.

4.145 erf

Calculate erf(x)

Arguments

Number	Type	Compulsory	Default	Description
1	real array	yes		x

Returns

Return type: real array

erf(x)

Example

4.146 erfc

Calculate erfc(x)

Arguments

Number	Type	Compulsory	Default	Description
1	real array	yes		x

Returns

Return type: real array

erfc(x)

Example

4.147 EscapeString

Process string to replace escaped characters with literals.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		input string
2	string	Yes		options

Argument 1

Input string

Argument 2

Set to 'replacespaces' to enable \s which is substituted with a single space

Returns

Return type: string

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)
\s	Replaced with a single space. Enabled is arg2 set to 'replacespaces'. Can be used to create strings that contain no spaces.
\\	Replaced by a single '\'
'\' followed by any other character	Replaced by the character following the '\'. The '\' itself is omitted.

4.148 EscapeStringEncode

Process string and replace literals with escaped characters. Performs the reverse operation to [EscapeString\(\)](#)

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		input string

Argument 1

Input string

Returns

Return type: string

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

Literal value	Replaced with:
Tab character (ASCII code 9)	\t
New line character (ASCII code 10)	\n
Carriage return character (ASCII code 13)	\r
Form feed character (ASCII code 12)	\f
'\'	\\

4.149 *ev*

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

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

Arguments

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

Returns

Return type: real/complex array

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.

4.150 Execute

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](#).

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Script name
2	any	No		Script argument 1
3	any	No		... Upto 8 args in total

Argument 3

Script args 2-7

Returns

Return type: Depends on called script

4.151 ExistCommand

Test if a script command is a valid command.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Command name

Returns

Return type: real

Returns 1.0 if the command is available otherwise 0.0

Notes

There are two situations where a documented command may not be available:

- The command is not implemented in the currently executing version of the application.
- The command is not enabled with the current license. A few commands are 'licensed' and are not available with all products.

See Also

[ExistFunction\(\)](#)

4.152 ExistDir

Checks if the specified directory exists

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Directory name

Returns

Return type: real

Function returns a real scalar with one of three values:

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

4.153 ExistFile

Tests whether the given file exists. Does not test whether the file can be opened. Use [CanOpenFile\(\)](#) to test if a file exists and can also be opened.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File path

Returns

Return type: real

1.0 if file exists otherwise 0.0

4.154 ExistFunction

Returns TRUE or FALSE depending on whether specified function exists.

Arguments

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

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 command [RegisterUserFunction](#).

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

Returns

Return type: real

Notes

There are two situations where a documented function may not be available:

- The function is not implemented in the currently executing version of the application.
- The function is not enabled with the current license. A few functions are 'licensed' and are not available with all products.

4.155 ExistSymbol

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Symbol name
2	string	No	'global'	Scope

Returns

Return type: real

4.156 ExistVec

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](#) for further details.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		vector name
2	string	No	'global'	options

Returns

Return type: real

4.157 exp

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

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

The exponential of the argument.

4.158 fd

Alias of FinalDivision [FinalDivision\(\)](#)

Arguments

Number	Type	Compulsory	Default	Description
1	Real/complex	Yes		Input vector

Argument 1

Input vector

Returns

Return type: Same as arg 1

Final division of input vector

4.159 *fft*

Performs a Fast Fourier Transform on supplied vector. The number of points used is the next binary power higher than the length of argument 1.

Arguments

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

Argument 2

Values are either 'Hanning' (default) or 'None'.

Returns

Return type: complex array

Notes

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. 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.

4.160 *Field*

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 in the Prop command](#).

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		value
2	real	Yes		first bit
3	real	Yes		second bit

Returns

Return type: real

4.161 FileToString

Returns the entire contents of a file as a single string

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File name
2	string	No	'auto'	Encoding
3	string	No	Function returns an empty vector on failure if this argument is not supplied	Default output in the event of failure

Argument 2

Character encoding assumed for input file. May be any value returned by the function [GetCodecNames\(\)](#). Examples include:

'utf-8' UTF8 encoding. This is the encoding used internally and for output

'utf-16' UTF16 also known as UCS-2

'Shift-JIS' Commonly used on Japanese systems

In addition the following special values may be used:

'locale' uses the default encoding for the system's locale

'auto' uses 'utf-8' if successful. Otherwise uses 'locale'

Argument 3

Default output in the event of failure

Returns

Return type: String

String containing contents of file

4.162 FilterEditMenu

Filters a menu list to return only menu definitions that are actually displayed.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Menu definition list

Argument 1

The menu definition list, as given by global:menusnapshot.

Returns

Return type: string array

Same as the input, but with entries removed for menus that are not displayed but rather form menus that are built up.

4.163 FilterFile

***** UNSUPPORTED ***** – See page 26 for more information

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		file name
2	string array	Yes		keywords
3	string	No		option

Returns

Return type: string array

4.164 FinalDivision

Returns final division of a multi-division vector. If the vector is single division the function will simply return the vector

Arguments

Number	Type	Compulsory	Default	Description
1	Real/complex	Yes		Input vector

Argument 1

Input vector

Returns

Return type: Same as arg 1

Final division of input vector

4.165 FindGraphMeasurement

Returns graph measurement objects that match a provided curve ID and, optionally, a Label value

Arguments

Number	Type	Compulsory	Default	Description
1	String Array	Yes		Curve ID
2	String	No	Matches any label	Measurement label

Argument 1

ID of curve associated with measurement objects

Argument 2

Measurement label

Returns

Return type: String array

IDs of measurement objects that are associated with the provided curve ID. If the Label value is provided, only those objects matching the label will be returned

See Also

[CreateGraphMeasurement\(\)](#)

[EditGraphMeasurement\(\)](#)

4.166 FindModel

Returns the file path and line number of a simulator model given its name and type

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Model name
2	string	Yes		Model letter
3	string	No	'SIMetrix'	Simulator type

Argument 1

Model name, this is either the name in a .MODEL statement or the name in a .SUBCKT statement.

Argument 2

Model letter, e.g 'Q' for BJTs, 'D' for diodes and 'X' for subcircuits.

Argument 3

Simulator type, i.e 'SIMetrix' or 'SIMPLIS'

Returns

Return type: String array

String array of length 2 holding the file name and line number of the definition of the specified model.

4.167 FIR

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. 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. A 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.

Arguments

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

Returns

Return type: real array

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))
```

See Also

[IIR\(\)](#)

4.168 Floor

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\(\)](#) for a version that accepts vector input.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		scalar

Returns

Return type: real

4.169 floorv

Returns the argument truncated to the next lowest integer. Same as `Floor()`, except that it also accepts vector inputs, for example:

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

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		vector

Returns

Return type: real vector

Returns a vector of the arguments truncated to next lowest integers

4.170 FormatNumber

Formats a real value and returns a string representation of it.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		number
2	real	Yes		significant digits
3	string	No	'eng'	format

Argument 3

Format options are:

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

Returns

Return type: real

4.171 Fourier

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		data
2	real	Yes		Fundamental frequency
3	real	Yes		Number of frequency terms
4	real array	No		options

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.

Returns

Return type: complex array

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

4.172 FourierOptionsDialog

Same as [DefineFourierDialog\(\)](#) except that only the Fourier sheet is displayed. The remaining tabbed sheets are hidden.

Arguments

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

Returns

Return type: string array

4.173 FourierWindow

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		input vector
2	string	No	'hanning'	window type

Argument 1

Input vector

Argument 2

Window type. One of:

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

Returns

Return type:

4.174 FullPath

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

Arguments

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

Returns

Return type: real

Example

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

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

See Also

[RelativePath\(\)](#)

[SplitPath\(\)](#)

4.175 *gamma*

Calculate $\gamma(x)$

Arguments

Number	Type	Compulsory	Default	Description
1	real array	yes		x

Returns

Return type: real array

$\gamma(x)$

Example

4.176 *Gauss*

Returns a random number with a Gaussian distribution. This function is intended to be used for SIMPLIS Monte Carlo analyses and would typically be used in device value expressions.

This function is only available in the Simulator process and cannot be called from scripts running in the context of the front end. The function is only active when used by the netlist pre-processor with Monte Carlo analysis enabled. When used in other contexts, the function returns 1.0.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Tolerance - 3-sigma spread

Returns

Return type: real

Random number with a Gaussian distribution of mean 1.0 and standard deviation of tolerance/3 where tolerance is the value supplied to argument 1.

Returns 1.0 when used in non Monte Carlo contexts.

Example

1k*Gauss(0.1) will return 1000 +/- 10% with a 3-sigma spread. Returns 1.0 in a non Monte Carlo run.

Notes

The function can return values outside the tolerance range. For example Gauss(0.1) can return values greater than 1.1 and less than 0.9 which would violate the tolerance specification for many components. Use the [GaussTrunc\(\)](#) function to get a distribution that does not extend beyond the tolerance range.

See Also

[Unif\(\)](#)

[GaussTrunc\(\)](#)

[Distribution\(\)](#) - also alias [UD\(\)](#)

[WC\(\)](#)

[WC2\(\)](#)

4.177 GaussLim

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Tolerance
2	real	No	3	sigma multiplier

Returns

Return type: real

4.178 GaussTrunc

Returns a random number with a Gaussian distribution but truncated so that it won't return values outside the specified tolerance range. This function is intended to be used for SIMPLIS Monte Carlo analyses and would typically be used in device value expressions.

This function is only available in the Simulator process and cannot be called from scripts running in the context of the front end. The function is only active when used by the netlist pre-processor with Monte Carlo analysis enabled. When used in other contexts, the function returns 1.0.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Tolerance
2	real	No	3	sigma multiplier

Returns

Return type: real

Random number with a Gaussian distribution of mean 1.0 and standard deviation of (tolerance/sigma_multiplier) where tolerance is the value supplied to argument 1 and sigma_multiplier is the argument provided to argument 2. Values outside the range 1.0 +/-tolerance are rejected so the function will never return values outside this range

Example

1k*GaussTrunc(0.1) will return 1000 +/- 10% with a 3-sigma spread. Will not return values outside the range 0.9-1.1. Returns 1.0 in a non Monte Carlo run.

See Also

[Gauss\(\)](#)

[Unif\(\)](#)

[Distribution\(\)](#) - also alias [UD\(\)](#)

[WC\(\)](#)

4.179 GenPrintDialog

Opens a dialog box used to define print settings

Arguments

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

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	<ul style="list-style-type: none"> '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: <ul style="list-style-type: none"> '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: <ul style="list-style-type: none"> 'on' Checked 'off' Not checked
12	Minor grid checked: <ul style="list-style-type: none"> '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'.

Returns

Return type: string array

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.

4.180 GetActualPath

Returns actual file or directory path as a full path even if the path passed is a symbolic or hard link. If the path is a network share it will return a server style UNC path. It will also convert "8.3" short paths to "long" paths. Path returned will always use native path separators (i.e. backslashes) but will accept forward slashes on input. This will return an empty string if the object pointed to does not exist or cannot be opened

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path

Returns

Return type: string

Resolved path

4.181 GetAllAxes

Returns an array listing all axis ids for currently selected graph. Note that by default, digital axes are excluded.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Options

Argument 1

Set to 'Dig' to include digital axes in the result

Returns

Return type: string array

4.182 GetAllCurves

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.

Arguments

No arguments

Returns

Return type: string array

4.183 GetAllSimulatorDevices

Returns a list of semi-colon delimited strings containing information on all built-in simulator devices.

Arguments

No arguments

Returns

Return type: string array

Array of semi-colon delimited strings. The strings in the field are defined in the following table:

Field	Description
0	Device name
1	Model name - as used in the .MODEL statement. E.g npn, nmos etc.
2	Level parameter value
3	Minimum number of terminals
4	Maximum number of terminals
5	Device letter. E.g. 'Q' for BJTs, 'D' for diodes

Example

4.184 GetAllSymbolPropertyNames

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Options

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.

Returns

Return type: string array

4.185 GetAllXAxes

Returns an array listing all x axis ids for currently selected graph. Note that by default, digital axes are excluded.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Options

Argument 1

Set to 'Dig' to include digital axes in the result

Returns

Return type: string array

4.186 GetAllYAxes

Returns an array listing all y axis ids for currently selected graph. Note that by default, digital axes are excluded.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Options

Argument 1

Set to 'Dig' to include digital axes in the result

Returns

Return type: string array

4.187 **GetAnalysisInfo**

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( (GetAnalysisInfo())[stopIdx])
```

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Options

Argument 1

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, 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
Operating point	ANALYSISNAME, GROUPNAME

Returns

Return type: string array

4.188 **GetAnalysisLines**

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.

Arguments

No arguments

Returns

Return type: string array

4.189 GetAnnotationText

Returns the text of the requested annotation. This work for text annotations and shape annotations with text applied to them.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	'Ref'	Handle of the annotation

Returns

Return type: string array

The text of the requested annotation.

4.190 GetAxisCurves

Returns an array listing all curve id's for specified x or 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Y axis id

Returns

Return type: string array

Curve ids for curves attached to specified axis

Notes

Compatibility: In version 8.1 and earlier, GetAxisCurves returns an empty vector if the specified axis is an x-axis.

4.191 GetAxisLimits

Returns min and max limits and axis type (log or lin) of specified axis

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Axis ID

Returns

Return type: real array

Returns array of length 3 providing limits info for specified axis.

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

4.192 GetAxisType

This function is obsolete and should not be used for new code. To obtain the type of an axis (either X or Y) read the 'AxisType' property using [GetGraphObjPropValue\(\)](#).

The concept of a 'Main' axis is not obsolete. In versions 8.1 and earlier the grid at the bottom of the graph was described as the 'Main axis' or 'Main grid'. Other grids were described as just 'Grids'. The main axis or grid could have multiple y-axes whereas the other grids supported just a single y-axis. This limitation no longer applies and all grids have equal functionality. So there is no longer a concept of a main axis. However, for compatibility with old code, this function will return 'Main' for the grid at the bottom of the graph.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Axis ID

Returns

Return type: string

Returns the type of axis. Possible values are:

'X'	X-axis
'Digital'	A Digital Y-axis. (Created with <code>Curve /dig</code> 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

4.193 GetAxisUnits

Returns physical units of axis. See the function [Units\(\)](#) for list of possible values.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Axis ID

Returns

Return type: string

4.194 GetChildModulePorts

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.

Property name and value must uniquely define an instance.

Arguments

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

Argument 1

Usually arg 1 the property name is ‘handle’. If arg 1 is an empty string, a single selected instance will be used.

Argument 2

The property value

Argument 3

Schematic ID as returned by the [OpenSchematic\(\)](#) function. 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.

Returns

Return type: string array

String array of size = 2 times the number of module ports in the underlying schematic. Values arranged in pairs. The first in each pair is 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.

4.195 GetCodecNames

Returns all encoding types available to be used with [LoadFile\(\)](#), [SetDefaultEncoding](#) and all text editor open commands using the /encoding switch.

Arguments

No arguments

Returns

Return type: string array

4.196 GetColours

Returns the names of built-in colour objects.

Arguments

No arguments

Returns

Return type: string array

4.197 GetColourSpec

Returns the current colour specification for a colour object whose name is passed to argument 1. Named colour objects are simply option variables used to store colour information. See [Set](#) for information about option variables.

Returns the value in the form #rrggbb.

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Colour name

Returns

Return type: string

4.198 GetCompatiblePathName

Returns a "short" path name if the supplied path has white space or non-ascii characters. This function may not function as desired on all systems as not all file systems support short path names.

The function only replaces the parts of the path that have spaces or non-ASCII characters.

A short path is one that complies with the DOS 8.3 naming convention.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path

Argument 1

Input path. Maybe full or partial and the function will return its argument in the same form (that is, it won't convert to a full path). If the input path does not exist, this function will simply return its argument unmodified.

Returns

Return type: string

See Also

[GetLongPathName\(\)](#)

4.199 GetComponentValue

Same as [SetComponentValue\(\)](#) except that it can only read values. Refer to [SetComponentValue\(\)](#) for full details.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Address
2	real	No	-1	Schematic ID

Argument 2

Schematic ID as returned by the [OpenSchematic\(\)](#) 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.

Returns

Return type: string array

Refer to [SetComponentValue\(\)](#) for details

4.200 GetConfigLoc

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 of the registry key is `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.

Arguments

No arguments

Returns

Return type: string array

4.201 GetConnectedPins

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		schematic location
2	string	No	'ref'	identifying property
3	string	No	'pinname'	pin number of pin name

Argument 1

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

Returns

Return type: string array

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]])

```

4.202 GetConvergenceDevNames

List device references that have convergence info available. Requires ".OPTIONS advConvReport" to be specified in the simulation netlist

Arguments

No arguments

Returns

Return type: String array

Array of device references

4.203 GetConvergenceInfo

Return convergence data for most recent simulation

Arguments

No arguments

Returns

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.

4.204 **GetConvergenceNodeNames**

List node names that have convergence info available. Requires ".OPTIONS advConvReport" to be specified in the simulation netlist

Arguments

No arguments

Returns

Return type: String array

Array of nodenames

4.205 **GetCurDir**

Returns current working directory.

Arguments

No arguments

Returns

Return type: string

Returns current working directory.

4.206 **GetCurrentGraph**

Returns id of the currently selected graph.

Arguments

No arguments

Returns

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.

See Also

[GetGraphObjPropValues\(\)](#)
[GetGraphObjPropValue\(\)](#)
[GetGraphObjects\(\)](#)
[GetGraphObjPropNames\(\)](#)
[GetSelectedGraphAnno\(\)](#)

4.207 GetCurrentStepValue

Returns the current step value in a script-based multi-step analysis. Script-based multi-step analyses use a script call to define each step. For this analysis type, a counter is maintained which increments on each step. This function returns the value of that counter. Note that the counter is initialised to 1.

Arguments

No arguments

Returns

Return type:

Example

The following script code sets the BF parameter to values of 100, 200 and 400 for the first, second and third steps respectively.

```

Let values = [100, 200, 400]
Let step = GetCurrentStepValue()
Let value = values[step-1]

Let SetModelParamValue('BC546B', 'BF', value)

```

See Also

[SetModelParamValue\(\)](#)
[SetInstanceParamValue\(\)](#)
[GetModelParameterValues\(\)](#)
[GetDotParamValue\(\)](#)

4.208 GetCursorCurve

Returns curve id and source group name of curve attached to measurement cursor

Arguments

No arguments

Returns

Return type: string array

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)

4.209 GetCurveAxes

Returns X and Y axis ids of specified curve as two-element array.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Curve ID

Returns

Return type: string array

Returns a two-element array containing the X-axis id in the first element and the Y-axis id in the second element. An empty vector will be returned if the specified curve-id is not a valid curve

4.210 GetCurveAxis

Returns y-axis id of specified curve.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Curve ID

Returns

Return type: string

Returns the id of the y-axis to which the specified curve is attached. If the object specified is not a curve, the function will return an empty string.

4.211 GetCurveCreationIds

The commands [Plot](#), [Curve](#) and [NewPlot](#) usually create new graph objects. This function returns the IDs of those objects.

Arguments

No arguments

Returns

Return type: Real array

Five element real array containing the IDs of the following objects

- 0: Graph ID
- 1: Grid ID
- 2: X-Axis ID
- 3: Y-Axis ID
- 4: Curve ID

If no [Plot](#), [Curve](#) or [NewPlot](#) command has been called, or the most recent call failed, the return values from this function will all be zero.

4.212 GetCurveName

Returns the label of specified curve. This is the `Label` property and is displayed in the legend window.

Arguments

Number	Type	Compulsory	Default	Description
1	Integer	Yes		ID of the curve

Returns

Return type: string

Returns name of specified curve.

4.213 GetCurves

Returns curve names in selected graph. The curve name is the curve's `Name` property.

Arguments

No arguments

Returns

Return type: string array

Returns an array of curve names for the current graph.

4.214 GetCurveVector

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](#). 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 this function and is convenient in situations where a short expression is desirable.

Arguments

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

Returns

Return type: real array

4.215 GetDatumCurve

Returns curve id and source group name of curve attached to reference cursor.

Arguments

No arguments

Returns

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)

4.216 GetDeviceDefinition

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Device name
2	string	Yes		Device type
3	string	No	'SIMetrix'	Simulator type
4	string	No		Options

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.

Returns

Return type: string array

4.217 GetDeviceInfo

Returns information about the specified simulator device.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Model name
2	string	No	none	Options

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.

Returns

Return type: string array

Result is a 7 element array about the specified simulator device.

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
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. 'npn', 'pnp' and 'lpnp' are returned for the 'BJT' device.

4.218 GetDeviceParameterNames

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Device type
2	real	No	-1	Level
3	string array	No		Options

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\(\)](#). 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\(\)](#).

Returns

Return type: string array

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\(\)](#).

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'
```

4.219 GetDevicePins

Returns information about the electrical connections on a specified simulator device

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		device identifier
2	string array	No		options

Argument 1

Device identifier. If 'instname' is specified in argument 2, this will be the instance reference of the device. Otherwise the device name must be specified.

Argument 2

Can be a combination of 'instname' and 'getterms'. 'instname' means use the instance name to define the device. 'getterms' is functional for Verilog-HDL devices and will instruct the function to return information on vectored terminals.

Returns

Return type: string array

Array of semi-colon delimited strings providing the following information about the electrical connections to the specified simulator device.

Field index	Description
1	Pin name
2	Direction - in, out or inout. Currently only Verilog-HDL devices will return anything other than in or out for this field
3	Discipline - Verilog-A devices will return the defined discipline for the connection. This field will be empty for other devices
4	Connection size for vector connections - Currently only Verilog-HDL devices will return anything other than 1 for this field

4.220 GetDeviceStats

Get simulation statistics for each device type

Arguments

No arguments

Returns

Return type: string array

Array of strings with each element containing a list of name=value pairs providing information on each device type used in the simulator. Information provided is as follows:

Name	Value
(unlabelled)	Device type
Tload	Time in seconds used to evaluate the device's equations. This entry will be zero unless '.OPTIONS devacct' is specified in the simulation netlist
Count	Number of instances of this device type
ByteCount	Number of bytes used to store the data for instances of this device

4.221 GetDisabledInstances

Returns a list of all disabled instances on the current schematic. Instances may be disabled using the [SetDisable](#) command. A disabled instance is inactive as if it were not present in the schematic.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Property used to report instances

Argument 1

Property used to report instances

Returns

Return type: String array

List of property values (defined by argument 1) identifying the disabled instances

4.222 GetDivisionLabels

Get the division labels for a specified graph curve. Division labels are assigned to multi-step data and described the name and value of the item that was swept.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Curve id

Argument 1

Curve id

Returns

Return type: String array

List of values usually in the form name=value

4.223 GetDotFilePath

Retrieves the temporary file used to implement a .FILE statement in a netlist. The .FILE/.ENDF statements allow a file to be embedded in a netlist. This can be used for a number of purposes. The .FILE/.ENDF statements work by writing their contents to a temporary file and the appropriate function then accesses that file. This function returns the pathname for the temporary file. It will continue to be valid after a simulation has completed but will be deleted when a new simulation is started

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		.FILE name

Argument 1

Name after .FILE statement

Returns

Return type:

Full path name of temporary file used to implement .FILE

4.224 GetDotParamNames

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

Arguments

No arguments

Returns

Return type: string array

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.

4.225 GetDotParamValue

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Variable name

Returns

Return type: real

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

4.226 GetDriveType

Determines the type of drive or file system of the specified path.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path

Returns

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
'notexist'	The path doesn't exist or media not present
'unknown'	Drive type or file system could not be determined

4.227 GetEmbeddedFileName

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File path

Argument 1

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

Returns

Return type: string

'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 that script.

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

4.228 GetEnvVar

Returns the value of a system environment variable.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		System environment variable name

Returns

Return type: string

4.229 GetEthernetAddresses

Returns information about the installed Ethernet adapters.

Arguments

No arguments

Returns

Return type: string array

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.

4.230 GetF11Lines

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Options
2	real	No	-1	Schematic ID

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

Returns

Return type: string array

4.231 GetFailedNodesFromIterNum

Get list of failed nodes given an iteration value

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		

Argument 1

Iteration number

Returns

Return type: String array

List of node names

4.232 GetFile

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\(\)](#) and [GetUserFile\(\)](#) which are more flexible.

Arguments

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

Returns

Return type: string

4.233 GetFileCD

This function is now obsolete. Use the functions [GetSimetrixFile\(\)](#) or [GetUserFile\(\)](#) instead.

Arguments

No arguments

Returns

Return type:

4.234 GetFileDir

Get the directory where the specified file is located.

The function first converts the supplied path to a full path then strips off the final component of the path. If the path actually points to a directory, the value returned will be the parent directory. The function does not check that the path supplied actually exists.

Arguments

Number	Type	Compulsory	Default	Description
1	string	yes		Path to file. May be a relative path

Returns

Return type:

Full directory path where file is located

4.235 GetFileExtensions

Returns a string array containing all valid extensions (without prefixed '.') for the given file type.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File type

Returns

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
'Data'	Data files	DataExtension	sxdat, dat
'Text'	Text files	TextExtension	txt, log
'Symbol'	Binary symbol files	SymbolExtension	sxslb, slb
'LogicDef'	Logic definition files used with arbitrary logic block	LogicDefExtension	ldf
'Script'	Script files	ScriptExtension	sxscr
'Model'	Model files	ModelExtension	lb, lib, mod, cir, spi, fam, mdl, sp, sp2, model, pkg, prm, sub, sio, ckt
'Catalog'	Catalog files	CatalogExtension	cat
'Graph'	Graph binary files	GraphExtension	sxgph
'Component'	Schematic hierarchical component	ComponentExtension	sxcmp
'Snapshot'	Snapshot files	SnapshotExtension	sxsnp
'Netlist'	Netlist files	NetlistExtension	net, cir, deck
'Verilog-A'	Verilog-A files	VerilogAExtension	va, vams
'Verilog-HDL'	Verilog-HDL files	VerilogHDLExtension	v

Argument	Used for	Option name	Default
'ASCIIFileEditor'	Schematic ASCII files	AsciiFileEditorExtension	sxsch, sxsbl, sxcmp

You can combine multiple file types delimited by '&'. For example "Netlist & Model" will return the extensions for both netlist and model file types.

4.236 GetFileInfo

Returns information about a specified file.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File path

Returns

Return type: string array

Returns an array of length 5.

Index	Description
0	Drive type, one of: 'local', 'cdrom', 'remote', 'other', 'notexist', 'unknown'. See notes for function GetDriveType() .
1	File size in bytes
2	Full path name
3	Last modified time. Value is the number of seconds elapsed since January 1, 1970.
4	'True' if path is a directory, otherwise 'false'

4.237 GetFileSave

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

Arguments

No arguments

Returns

Return type:

4.238 GetFileVersionStamp

Returns file version stamp

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File path
2	string	No		Options

Argument 1

File path

Argument 2

If set to 'usestringinfo' the FileVersion string will be read instead of the integer values. Set this if you need the behaviour of this function to be the same as SIMetrix version 7.2 or earlier.

Returns

Return type: string

Version stamp typically in form major.minor.service.build

4.239 GetFileViewerSelectedDirectories

Returns the full path names of directories selected in all of the File Views.

Arguments

No arguments

Returns

Return type: string array

List of path names, each array item is a separate path name.

See Also

[GetFileViewerSelectedFiles\(\)](#)

[AddFileViewMenuItem](#)

4.240 GetFileViewerSelectedFiles

Returns the full path names of files selected in all of the File Views.

Arguments

No arguments

Returns

Return type: string array

List of path names, each array item is a separate path name.

See Also

[GetFileViewerSelectedDirectories\(\)](#)

[AddFileViewMenuItem](#)

4.241 GetFirstSelectedElementOfType

Returns handle of first selected schematic element of the requested type or types.

If multiple types are given, a search will be conducted on each type in turn, until a selected element of one of the requesting types is found. Only one handle is returned and this is the first element that the search comes across that is selected and is of the type requested.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Element type or types

Argument 1

Either a single element type, or an array of different types. If several types are provided, it will search for a selected element of the different types in order, meaning that if there is a match for the first array index, any subsequent indexes will not be searched.

Available elements types are:

ArrowAnnotation

ImageAnnotation

Instance

LineAnnotation

ShapeAnnotation

TextAnnotation

TitleBlock

Wire

Returns

Return type: string

Handle of the first selected element of the type requested, or an empty string if no matching elements were found.

4.242 GetFonts

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		options

Argument 1

If set to 'supportcolour' will return only fonts that have an editable colour.

Returns

Return type: string array

4.243 GetFontSpec

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\(\)](#) function and [EditFont](#) command.

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

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Object name

Returns

Return type: string

string

4.244 GetFreeDiskSpace

Returns free space on disk volume holding specified file or directory.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		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:'

Returns

Return type: real

Free space available in bytes

4.245 GetGraphFromWindow

Returns a graph ID given a window ID.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Window ID

Argument 1

Window ID as returned by one of these functions:

[WM_GetContentWidgetNames](#)

[WM_GetContentWidgetsOfType](#)

[WM_GetAllVisibleContentWidgetNames](#)

[WM_GetLastAccessedContentWidget](#)

Returns

Return type: String

Graph ID

4.246 GetGraphObjects

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 for all graph sheets are returned.

If the first argument is specified, all objects of the defined type for all graph sheets 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.

See [Graph Objects](#) for information on graph objects.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Object type name
2	string	No	Current graph	Graph ID
3	string	No		Options

Argument 3

Set to 'selected' to return selected objects only

Returns

Return type: string array

See Also

[GetGraphObjPropValues\(\)](#)

[GetGraphObjPropValue\(\)](#)

[GetCurrentGraph\(\)](#)

[GetGraphObjPropNames\(\)](#)

[GetSelectedGraphAnno\(\)](#)

4.247 GetGraphObjPropNames

Returns the valid property names for the graph object defined by argument 1. See [Graph Objects](#) for more information.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Graph object ID

Returns

Return type: string array

See Also

[GetGraphObjPropValues\(\)](#)

[GetGraphObjPropValue\(\)](#)

[GetGraphObjects\(\)](#)

[GetCurrentGraph\(\)](#)

[GetSelectedGraphAnno\(\)](#)

4.248 GetGraphObjPropValue

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\(\)](#).

If argument 3 is set to 'array', any array properties will be output as a string array. Otherwise the array values will be composed into a single string in the form:

[val1, val1, ...]

See [Graph Objects](#) for more information.

(Note the function [GetGraphObjPropValues](#) is the same but will only accept one argument)

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Graph object ID
2	string	No	Return all values	Property name
3	string	No		Options

Returns

Return type: string array

Index	Description
0	

See Also

[GetGraphObjPropValues\(\)](#)
[GetGraphObjects\(\)](#)
[GetCurrentGraph\(\)](#)
[GetGraphObjPropNames\(\)](#)
[GetSelectedGraphAnno\(\)](#)

4.249 GetGraphObjPropValues

Returns property values for the specified object. The function will return an array containing all property values. The order of the values corresponds to the return value of [GetGraphObjPropNames\(\)](#).

See [Graph Objects](#) for more information.

See [GetGraphObjPropValue\(\)](#) to obtain one property at a time.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Graph object ID

Returns

Return type: string array

Index	Description
0	

See Also

[GetGraphObjPropValue\(\)](#)
[GetGraphObjects\(\)](#)
[GetCurrentGraph\(\)](#)
[GetGraphObjPropNames\(\)](#)
[GetSelectedGraphAnno\(\)](#)

4.250 GetGraphTabs

Returns the graph IDs of all graphs currently open

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	N/A	No longer used

Argument 1

No longer used

Returns

Return type: string array

Returns an array of strings of length equal to the number of graphs currently open. 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\(\)](#) to obtain information about the graph including curves, axes, titles etc.

4.251 GetGraphTitle

Returns title of currently selected graph. This is the TitleBar property of the graph object.

Arguments

No arguments

Returns

Return type: string

4.252 GetGridAxes

Returns ids of all axes in the specified Grid. Grids are sections of a graph that contains one or more horizontally stacked Y-axes and one or more vertically stacked X-axes.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		

Argument 1

Grid id

Returns

Return type: string array

Curve ids

Notes

This function returns all axes both X and Y. These can also be obtained from the XAxes and YAxes properties of the Grid object. For example to get the y-axes use function call `GetGraphObjPropValue(ID, 'yaxes', 'array')`. See [GetGraphObjPropValue\(\)](#)

4.253 GetGridCurves

Returns ids of all curves in the specified Grid. Grids are sections of a graph that contains one or more horizontally stacked Y-axes and one or more vertically stacked X-axes.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		

Argument 1

Grid id

Returns

Return type: string array

Curve ids

Notes

The curves belonging to a specified grid may also be obtained by reading the Grid's Curves property using the function call: `GetGraphObjPropValue(ID, 'curves', 'array')`. See [GetGraphObjPropValue\(\)](#)

4.254 GetGroupAnalysisInfo

Retrieves analysis info stored in data groups.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	Current group	group name

Argument 1

group name

Returns

Return type: string array

Returns a string array with 7 elements:

- Analysis step mode
- Analysis type (e.g. tran, ac etc)
- Step type, None, Stepped or Statistical
- Appended group "true" or "false"
- Log sweep mode "true" or "false"
- Start value (e.g. start time)
- End value (e.g. end time)

4.255 GetGroupFromAnalysisId

Get group name given its analysis ID. The analysis ID can be added to some simulation commands using the ANALYSIS_ID parameter name. E.g.

```
.tran 1u 1m ANALYSIS_ID=tran_23
```

This function can be used to retrieve the name of the data group where the results were written. The ANALYSIS_ID may be used with both SIMPLIS and SIMetrix analysis lines. ANALYSIS_ID can be used with SIMetrix .TRAN, .DC, .AC, .NOISE and .TF analysis lines and .POP, .AC and .TRAN SIMPLIS analysis lines.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Analysis ID value
2	Real	No	Main group	Main or DCOP group. (SIMetrix only)

Argument 1

Analysis ID value

Argument 2

Boolean: 0: return name of main group, 1: return name of associated DC operating point group

Returns

Return type: String

Name of data group that carries data for analysis

4.256 GetGroupInfo

Returns information about a group.

For more information on groups, see [Groups](#).

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Group name

Argument 1

Group name for which information is required. Enter ‘’ to obtain information on the current group.

Returns

Return type: string array

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

4.257 GetGroupStepParameter

Returns the names of the ‘stepped parameters’ of a multi-step run. These values are stored within the group created for the simulation run’s output data. The stepped parameters are labels that identify the parameters, devices, model parameters or other quantities that are varied during a multi-step run.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	Current group	Group name

Returns

Return type: string array

4.258 GetGroupStepVals

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 parameters’ (see the function [GetGroupStepParameter\(\)](#)) during a multi-step run.

If there is more than one stepped parameter, the second argument may be used to identify for which parameter the values are returned.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	Current group	Group name
2	real	No	0	index

Argument 2

Identifies parameter when there is more than one

Returns

Return type: real array

4.259 GetGUID

Creates a GUID

Arguments

No arguments

Returns

Return type: String

GUID

4.260 GetHighlightedWidgetId

Returns ID of highlighted widget. In practice this always returns the same value as [WM_GetLastAccessedContentWidget](#)

Arguments

No arguments

Returns

Return type: string

ID of highlighted widget

4.261 GetHostId

Get MAC address or dongle serial numbers used for licensing

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	-1	Host id type

Argument 1

Can be the following value

Value	Description
'-1'	Default host id - this is the MAC address on Windows systems
'2'	MAC address
'15'	Serial number of FLEXid-9 type dongle
'51'	Serial number of FLEXid-10 type dongle

Returns

Return type: string

String as used in a license file

4.262 GetHttpContentSize

Get the size of an HTTP object specified by its URL. This function can be used to determine the size of a file on a web server. Note that for the function to work the HTTP server must send the required header information containing the file size. Some servers may not send the information.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		URL

Argument 1

URL of object whose size is requested

Returns

Return type: Real

Size of object in bytes. On failure (such as a an invalid URL) the function returns -1.

4.263 GetInstanceBounds

Returns the bounds occupied by a schematic instance identified by a property name and value.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name
2	string	Yes		Property value
3	string	No	none	Options

Argument 1

Property name to identify instance used in conjunction with parameter 2. 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 2

Property value to identify instance, along with parameter 1. 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 '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.

Returns

Return type: real array

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.

4.264 GetInstanceConvergenceInfo

Returns convergence information about a device. Requires ".OPTIONS advConvReport" to be specified in the simulation netlist.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Device reference
2	Real	No	0	Start index
3	Real	No		Output option

Argument 1

Device reference

Argument 2

Start index

Argument 3

If omitted, the return value is a string array providing all information available for the specified device. Otherwise an integer value may be specified which defined the item to be returned as a numeric array.

Returns

Return type: See notes for argument 3

See notes for argument 3

4.265 GetInstanceParamValues

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Instance name
2	string	No	Get all parameters	Parameter name
3	string	No		Options

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\(\)](#).

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.

Returns

Return type: string or string array

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.

4.266 GetInstancePinLocs

Return an array of pin locations for the symbol identified by arguments 1 and 2.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Property name
2	string	No		Property value
3	string	No	'relative'	Options
4	real	No	-1	Schematic ID

Argument 1

Property name to identify instance. Along with parameter 2, 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 2

Property value to identify instance. Along with parameter 1, 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 function [InstPoints\(\)](#).

Argument 4

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: real array

4.267 GetInstsAtPoint

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 times 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*).

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Instance pin location
2	string	Yes		Property name

Argument 1

specifies the pin location and is the value returned from the [GetInstancePinLocs\(\)](#) with the 'absolute' option specified.

Returns

Return type: real array

4.268 GetInternalDeviceName

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\(\)](#) - require the internal device name as an argument.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Model details

Returns

Return type: string array

1 - 3 element string array which describes device.

Index	Description
0	Model type name as used in the .MODEL control. E.g. 'nmos', 'nnp' etc.
1	Optional. Value of LEVEL parameter. If omitted, default level is assumed.
2	Optional. Value of VERSION parameter.

4.269 GetKeyDefs

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.

Arguments

No arguments

Returns

Return type: string array

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:

Index	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.

4.270 GetKnownFolderPath

Get system path location as defined by the operating system.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		String to identify folder to return

Argument 1

String to identify folder to return. The following table lists all the strings currently recognised. Not all of these will return a path on all systems and many are probably not meaningful.

For more information on these values, search the Internet for Microsoft documentation on KNOWNFOLDERID. Information for each will be listed under FOLDERID_XXX. For example see FOLDERID_Downloads for information on the folder returned for `GetKnownFolderPath('Downloads')`

AdminTools	Pictures	SamplePictures	ConflictFolder
CDBurning	PrintHood	SampleVideos	ConnectionsFolder
CommonAdminTools	Profile	SavedGames	ControlPanelFolder
CommonPrograms	ProgramData	SavedSearches	Games
CommonStartMenu	ProgramFiles	SendTo	HomeGroup
CommonStartup	ProgramFilesX64	SidebarDefaultParts	HomeGroupCurrentUser
CommonTemplates	ProgramFilesX86	SidebarParts	InternetFolder
Contacts	ProgramFilesCommon	StartMenu	LocalizedResourcesDir
Cookies	ProgramFilesCommonX64	Startup	NetworkFolder
Desktop	ProgramFilesCommonX86	System	OriginalImages
DeviceMetadataStore	Programs	SystemX86	PhotoAlbums
Documents	Public	Templates	Playlists
DocumentsLibrary	PublicDesktop	UserPinned	PrintersFolder
Downloads	PublicDocuments	UserProfiles	PublicUserTiles
Favorites	PublicDownloads	UserProgramFiles	RecycleBinFolder
Fonts	PublicGameTasks	UserProgramFilesCommon	RoamedTileImages
GameTasks	PublicLibraries	Videos	RoamingTiles
History	PublicMusic	VideosLibrary	SamplePlaylists
ImplicitAppShortcuts	PublicPictures	Windows	Screenshots
InternetCache	PublicRingtones		SEARCH_CSC

Libraries	PublicVideos	AccountPictures	SearchHome
Links	QuickLaunch	AddNewPrograms	SEARCH_MAPI
LocalAppData	Recent	ApplicationShortcuts	SyncManagerFolder
LocalAppDataLow	RecordedTVLibrary	AppsFolder	SyncResultsFolder
Music	ResourceDir	AppUpdates	SyncSetupFolder
MusicLibrary	Ringtones	ChangeRemovePrograms	UsersFiles
NetHood	RoamingAppData	CommonOEMLinks	UsersLibraries
PicturesLibrary	SampleMusic	ComputerFolder	

Returns

Return type: string

Full path of specified location

4.271 GetLaplaceErrorMessage

The function [ParseLaplace\(\)](#) returns a status code in the first field of its return value. This function converts it to an error message.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		code

Returns

Return type: string

Error message

4.272 GetLastCommand

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”.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Command group

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.

Returns

Return type: string

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.

4.273 GetLastError

Returns a string with one of three values signifying the status of the most recent command executed.

The command switches `/noerr` and `/quiet` (see [Command Switches](#)) 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'.

Arguments

No arguments

Returns

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.

4.274 GetLastGraphObjectAdded

Returns ID of last graph object added. The type of object can also be specified, using one of the object type names, where the last object of that type will be returned.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Object type

Argument 1

The type of object to search for the last added element of.

Returns

Return type:

Index	Description
0	

4.275 GetLegendProperties

Returns measurement names or values. These were called ‘Legend properties’ in versions 8.1 and earlier, hence the name of the function. From version 8.2, measurements are displayed in the measurement window and are graph objects with an id and properties. Although the underlying implementation of measurements has radically changed, this function nevertheless remains compatible with older versions, and is fully supported.

If argument 2 = ‘values’ the function returns legend property values. Otherwise it returns legend property names.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Curve ID
2	string	No	‘names’	Options

Returns

Return type: string array

Notes

If the measurement refers to a multi-division curve, that is one created from a multi-step analysis such as Monte Carlo, the value returned by this function will be the result of the first division only. This is compatible with the behaviour of version 8.1 and earlier. However, values for all divisions are calculated and available. These can be obtained by interrogating the properties of the measurement object. Use `FindGraphMeasurement` to obtain the measurement ID then read the ‘Value’ property using `GetGraphObjPropValue(ID, ‘value’, ‘array’)`.

See Also

[FindGraphMeasurement\(\)](#)

[GetGraphObjPropValue\(\)](#)

4.276 GetLibraryModels

Returns a string array containing information about each model in the specified model library.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Library spec
2	string	No		Options

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\(\)](#).

Argument 2

If set to 'usermodelsonly' only models installed by the user will be returned.

Returns

Return type: string array

String array with each element describing a single library model. Information is supplied as a semi-colon delimited string with the following fields:

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

4.277 GetLicenseInfo

Returns information about the current license.

Arguments

No arguments

Returns

Return type: string array

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

4.278 GetLicenseStats

Returns information about the license checkout process. This function is typically used to provide diagnostic information when a license checkout fails.

Arguments

No arguments

Returns

Return type: string array

Returns an array of strings. Each entry provides details of each license location. The first entry is always the license path for license files. 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.

4.279 GetLine

Returns a single line from a file.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		File handle

Argument 1

Handle as returned by the function [OpenFile\(\)](#).

Returns

Return type: string

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.

4.280 GetListSelected

Return list of selected elements from the ListSubsetDialog.

Argument list will be in the form: [selected] <elements>[notselected] <elements>.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		List elements

Returns

Return type: string array

Selected elements.

4.281 GetListUnselected

Return list of unselected elements from the ListSubsetDialog.

Argument list will be in the form: [selected] <elements>[notselected] <elements>.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		List elements

Returns

Return type: string array

Unselected elements.

4.282 GetLongPathName

Returns long path name for path specified either as a long or short path. Short path names are a feature of some file systems which represent the path in a form that would be accepted on legacy files systems especially DOS.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path

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.

Returns

Return type: string array

See Also

[GetShortPathName\(\)](#)

4.283 GetMaxCores

Return maximum cores available taking account of hardware capability and license

Arguments

No arguments

Returns

Return type: real

Maximum cores available

4.284 GetMD5

Get MD5 hash for a file

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Path to file

Argument 1

Full path to file to analyse

Returns

Return type: String

MD5 hash of specified file

4.285 GetMD5String

Get MD5 hash for a string

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Input string

Argument 1

Input string

Returns

Return type: String

MD5 hash of input string

4.286 GetMenuItems

Returns all menu item names in the specified menu.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Menu path
2	string	No		Options

Argument 1

Specifies the path for the menu as it would be provided to the command `DefMenu` 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.

Returns

Return type: string array

Returns a string array listing all the menu item names.

Example

```
GetMenuItems('Shell|&File')
```

returns all the menu items in the command shell's File menu.

4.287 GetModelFiles

Returns a list of currently installed device models.

Arguments

No arguments

Returns

Return type: string array

4.288 GetModelLibraryErrors

Returns list of error messages from model library install operations. List is cleared when this function is called.

Arguments

No arguments

Returns

Return type: string array

String array holding error messages

4.289 GetModelName

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'.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Instance name

Returns

Return type: string

4.290 GetModelParameterNames

Returns the names or default values of all real valued parameters for a device model.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Internal device name
2	string	No		default flag (unsupported)

Argument 1

Internal device name. This is returned by the functions [GetInternalDeviceName\(\)](#) and [GetModelType\(\)](#).

Argument 2

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.

Returns

Return type: string array

4.291 GetModelParameters

Returns the names and types of all parameters for a device model.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Internal device name
2	string	No	Use internal device name	options

Argument 1

Internal device name. This is returned by the functions [GetInternalDeviceName\(\)](#) and [GetModelType\(\)](#). If argument 2 is set to 'modelname' argument must be the model name of a model used in the most recent simulation

Argument 2

If set to 'modelname' argument 1 must be the name of a model used in the most recent simulation.

Returns

Return type: string array

String array of semi-colon delimited strings. Each token in the string is defined as follows:

Index	Description
0	parameter name
1	Parameter type
2	Parameter description - this is blank for most devices

4.292 GetModelParameterValues

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\(\)](#) for the same device and the values and parameter names map directly.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Model name
2	string	No	All values returned if omitted	Parameter name

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.

Returns

Return type: string array

4.293 GetModelType

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](#) 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Model name

Returns

Return type: string

4.294 GetModifiedStatus

Returns whether the specified schematic has been modified.

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	-1	Schematic ID

Argument 1

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: real

4.295 GetNamedSymbolPins

Returns the names for all pins of the specified symbol or hierarchical component.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Symbol name or component path
2	string	No	'symbol'	Options

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.

Returns

Return type: string array

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.

4.296 GetNamedSymbolPropNames

Returns names of all properties defined for a library symbol.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Internal symbol name
2	string	No	'symbol'	Options

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.

Returns

Return type: string array

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.

4.297 GetNamedSymbolPropValue

Returns the value of a property defined for a library symbol.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Internal symbol name
2	string	Yes		Property name
3	string	No	'symbol'	Options

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

Returns

Return type: string

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.

4.298 GetNearestNet

Returns information about the schematic net nearest the mouse cursor

Arguments

No arguments

Returns

Return type: string array

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'

4.299 GetNextDefaultStyleName

Returns next fully available default style name. This is used when creating new styles with a default name, where an index increments for additional styles created.

Names are in the form: MyStyleNormal[index] and MyStyleSelected[index], eg MyStyleNormal10.

Returns a name that will be valid for both the normal style and the selected style.

Arguments

No arguments

Returns

Return type: string

Next available style name that is not being used elsewhere, which can be used to create Normal and Selected variants of it.

4.300 GetNodeConvergenceInfo

Returns convergence information about a node. Requires ".OPTIONS advConvReport" to be specified in the simulation netlist.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Node name
2	Real	No	0	Start index
3	Real	No		Output option

Argument 1

Node name

Argument 2

Start index

Argument 3

If omitted, the return value is a string array providing all information available for the specified node. Otherwise an integer value may be specified which defined the item to be returned as a numeric array.

Returns

Return type: See notes for argument 3

See notes for argument 3

4.301 GetNodeNames

Returns all node names used in most recent simulation

Arguments

No arguments

Returns

Return type: string array

All node names used in simulation. Will return an empty vector if no simulation has been run

4.302 GetNonDefaultOptions

Returns names of all .OPTION settings in the most recent simulation that were not at their default value.

Arguments

No arguments

Returns

Return type: string array

4.303 GetNumCurves

Returns the number of curves in a curve group. This is applicable to curves plotted for a multi-step analyses such as Monte Carlo. Note that all non-digital curves plotted from multi-step analyses are grouped

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Curve ID

Returns

Return type: real

4.304 GetOpenSchematics

Returns the path names of all schematics currently open.

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	N/A	No longer used

Argument 1

No longer used

Returns

Return type: string array

A string array containing the full path names all schematics currently open.

See Also

[GetSchematicTabs\(\)](#) [SaveAs](#) [SelectSchematic](#)

4.305 *GetOption*

Returns the value of the *option variable* of name given as argument. *Option variables* are created using the command *Set* - see *User's Manual/Sundry Topics/Using the Set and Unset commands/List of Options* 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Option name

Returns

Return type: string

Index	Description
0	

4.306 *GetPath*

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	File system directory for the "My Documents" folder
ShareDir	The directory where the directories for symbol and model sub-directories are expected to reside. Typically <i>Program Files/SIMetrix80/support/</i> .

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Item name

Returns

Return type: string

4.307 GetPlatformFeatures

Returns information on availability of certain features that are platform dependent.

Arguments

No arguments

Returns

Return type: string array

Currently a string of length 4 defined as follows:

Index	Description
0	Is the function <code>ShellExecute()</code> implemented? 'true' or 'false'
1	Obsolete
2	Is 'VersionStamp' function implemented. 'true' or 'false'
3	Is context sensitive help implemented. 'true' or 'false'

4.308 GetPrinterInfo

Returns information on installed printers.

Arguments

No arguments

Returns

Return type: string array

Returns array of strings providing system printer names and current application default printer. 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	List of printer names (and subsequent indexes)


Example

The following is an example of executing the command `Show GetPrinterInfo`

```
Index  GetPrinterInfo()
0      '5'
1      '2'
2      'Dell Laser Printer 1100'
3      'Fax'
```



```

4 'HP Color LaserJet CP4020 Series PCL6'
5 'Microsoft XPS Document Writer'
6 'Send  OneNote 2010'

```

The default index is 2 so this means that 'HP Color LaserJet CP4020 Series PCL6' 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.

4.309 GetPrintValues

Returns the names of all quantities specified in .PRINT controls in the most recent simulation run.

Arguments

No arguments

Returns

Return type: string array

4.310 GetReadOnlyStatus

Returns the read only status of the specified schematic.

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	-1	Schematic ID

Argument 1

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: real

Returns 1.0 if the schematic is read-only. Otherwise returns 0.0

4.311 GetRegistryClassesRootKeys

List sub keys under key in registry HKEY_CLASSES_ROOT root

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		

Argument 1

Parent key path

Returns

Return type: string array

Sub keys under specified key

Example

4.312 GetSchematicFileVersion

Returns the file version for the requested schematic.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Schematic file path

Returns

Return type: string array

Version information about the file type.

Index	Description
0	Binary or ASCII file
1	Format version
2	Format revision

4.313 GetSchematicTabs

Returns IDs for all open schematics. The ID is an integer value that uniquely identifies a schematic and may be used by a number of commands and functions to perform operations on a schematic. For more information, refer to the [OpenSchematic\(\)](#) function.

Arguments

No arguments

Returns

Return type: real array

See Also

[GetOpenSchematics\(\)](#) returns the corresponding paths of open schematics [OpenSchematic\(\)](#) for more information on schematic IDs

4.314 GetSchematicVersion

Returns version information for the currently selected schematic.

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	-1	Schematic ID

Argument 1

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string array

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

4.315 GetSchemTitle

Returns the title of the current schematic.

Arguments

No arguments

Returns

Return type: string

4.316 GetSelectedAnnotationText

Returns the text in the selected annotation. Only works for a single selected annotation. If multiple annotations are selected, only the text from one of the annotations will be returned.

Arguments

No arguments

Returns

Return type: string

Annotation text

4.317 GetSelectedCurves

Returns the curve IDs of the selected curves.

Arguments

No arguments

Returns

Return type: string array

4.318 GetSelectedGraphAnno

Obsolete - for new code use [GetSelectedGraphAnnotations\(\)](#). Returns the ID for a single selected graph annotation object. If no objects are selected, the function returns '-1'. If no graphs are open, the function returns an empty vector.

If multiple objects are selected, the function will return the id of just one of them. To get all selected annotation objects, use the function [GetSelectedGraphAnnotations\(\)](#).

See [Graph Objects](#) for information on graph annotation objects.

Arguments

No arguments

Returns

Return type: string

Notes

Compatibility: Version 8.1 and earlier only allowed a single object to be selected so this function always returned a scalar value

See Also

[GetSelectedGraphAnnotations\(\)](#)

[GetGraphObjPropValues\(\)](#)

[GetGraphObjPropValue\(\)](#)

[GetGraphObjects\(\)](#)

[GetCurrentGraph\(\)](#)

[GetGraphObjPropNames\(\)](#)

4.319 GetSelectedGraphAnnotations

Returns the ID for all selected graph annotation objects. If no objects are selected, the function returns '-1'. If no graphs are open, the function returns an empty vector.

See [Graph Objects](#) for information on graph annotation objects.

Arguments

No arguments

Returns

Return type: string

See Also

[GetGraphObjPropValues\(\)](#)

[GetGraphObjPropValue\(\)](#)

[GetGraphObjects\(\)](#)

[GetCurrentGraph\(\)](#)

[GetGraphObjPropNames\(\)](#)

4.320 GetSelectedGrid

Returns id of selected Grid. Grids are sections of a graph that contains one or more horizontally stacked Y-axes and one or more vertically stacked X-axes.

Each axis is linked to a single grid which may be read from the Grid property.

Arguments

No arguments

Returns

Return type: string

4.321 GetSelectedStyleNames

Returns the names of the styles used by the selected elements. Each style name is returned at most once in the list.

Arguments

No arguments

Returns

Return type: string array

A list of style names that are used by the selected elements. For line and shape based elements that can be partially selected, the information is only returned if the element is fully selected.

4.322 GetSelectedXAxis

Returns id of selected x-axis.

Arguments

No arguments

Returns

Return type: string

4.323 GetSelectedYAxis

Returns id of selected y-axis.

Arguments

No arguments

Returns

Return type: string

4.324 GetShortPathName

Returns short path name for path specified either as a long or short path. Short path names are a feature of some file systems which represent the path in a form that would be accepted on legacy files systems especially DOS.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path

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.

Returns

Return type: string

See Also

[GetLongPathName\(\)](#)

4.325 GetSimConfigLoc

Returns the location of the simulator's configuration information. This function returns its result in an identical form to the function [GetConfigLoc\(\)](#).

Arguments

No arguments

Returns

Return type: string

Index	Description
0	

4.326 GetSimetrixFile

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File type
2	string array	No	<<empty>>	Options
3	string	No	<<empty>>	Initial file

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 files
'Snapshot'	Snapshot files
'Netlist'	Netlist files
'VerilogA'	Verilog-A files
'VerilogHDL'	Verilog-HDL files
'AsciiFileEditor'	Schematic ASCII files
'AnalysisData'	Monte Carlo, Sensitivity and Worst-case analysis files
'Optimiser'	Optimiser files

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...**

You can combine multiple file types delimited by '&'. For example "Netlist & Model" will select both netlist and model file types.

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.

Returns

Return type: string

4.327 GetSIMPLISExitCode

Returns the application exit code for the most recent SIMPLIS run. This may be used to determine whether SIMPLIS completed its run successfully.

Arguments

No arguments

Returns

Return type: real

Returns a single value according to the most recent SIMPLIS run.

4.328 GetSimulationErrors

Returns all errors raised by the most recent simulation.

Arguments

No arguments

Returns

Return type: string array

Returns a string array with all errors raised by the most recent simulation. If the simulation ran correctly with no errors, an **empty value** will be returned. Note that this function only returns error messages; it does not return warnings.

4.329 GetSimulationInfo

Returns information about the most recent simulation.

Arguments

No arguments

Returns

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	Always returns 'True'
3	Name of user specified data file
4	Not used - returns empty
5	.OPTIONS line specified at RUN command
6	Analysis line specified at RUN command
7	Reserved for future use
8	Netlist title. If run from the schematic editor, this will be the path of the schematic file.
9	Reserved for future use
10	Reserved for future use

4.330 GetSimulationSeeds

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.

Arguments

No arguments

Returns

Return type: real array

4.331 GetSimulatorEvents

***** UNSUPPORTED ***** – See page 26 for more information

Returns list of events for most recent simulation.

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.

Arguments

No arguments

Returns

Return type: string array

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:

Index	Description
0	See above table
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

4.332 GetSimulatorMode

Returns the simulator mode, that is SIMetrix or SIMPLIS, of the current schematic

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	-1	Schematic ID

Argument 1

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string

Return value may be ‘SIMetrix’ or ‘SIMPLIS’.

4.333 GetSimulatorOption

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. If no simulation has been run, this function will return an empty string.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Option name

Returns

Return type: string

4.334 GetSimulatorOptionInfo

Returns type and default value of a simulator option setting

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		option name

Returns

Return type: string array

Array of strings providing the following information

Index	Description
0	Option name
1	Type - can be 'REAL', 'INTEGER', 'BOOL', 'STRING' or 'UNKNOWN'
2	Default value

Notes

This function differs from [GetSimulatorOption\(\)](#) in that it returns information about an option setting independent of any simulation. [GetSimulatorOption\(\)](#) returns the value an option was set to in the most recent simulation.

See Also

[GetSimulatorOptions\(\)](#) [GetSimulatorOption\(\)](#)

4.335 GetSimulatorOptions

Return list of simulator options

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Output option

Argument 1

Can optionally be set to 'default' or 'type'. If set to 'default', function returns the default value instead of the name. If set to type, returns the type of the option - one of 'real', 'integer', 'boolean' or 'string'

Returns

Return type:

Array of strings holding names of all available options that can be set using the .OPTIONS statement. Optionally can return default value or type according to argument 2.

See Also

[GetSimulatorOptionInfo\(\)](#)

4.336 GetSimulatorStats

Returns statistical information about the most recent run

Arguments

No arguments

Returns

Return type: real array

Returns a 30 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
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.

4.337 **GetSimulatorStatus**

Returns the current status of the simulator.

Arguments

No arguments

Returns

Return type: string

May be one of the following values:

Value	Definition
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

4.338 GetSoaDefinitions

Returns all Safe Operating Area definitions specified in the most recent analysis.

Arguments

No arguments

Returns

Return type: string array

Returns an array of strings with each string in the form:

```
label;minvalue;maxvalue;xwindow;derating;type
```

Where:

label	The label specification on the .SETSOA line
minvalue	Minimum value
maxvalue	Maximum value
xwindow	Window width - the time the limits must be exceeded for the violation to be recorded
derating	Derating factor
type	'Peak' or 'Mean'

4.339 GetSoaMaxMinResults

Returns the maximum and minimum values reached for all SOA definitions.

Arguments

No arguments

Returns

Return type: string array

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:

<code>min_val</code>	Minimum value reached
<code>min_reached_at</code>	Time at which the minimum value was reached
<code>max_val</code>	Maximum value reached
<code>max_reached_at</code>	Time at which the maximum value was reached
<code>max_mean</code>	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.

4.340 GetSoaOverloadResults

Returns the overload factor for each SOA definition.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Options

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 $\frac{38}{40 \times 0.8} = 1.1875$. Without 'derated' specified, the overload factor would be $\frac{38}{40} = 0.95$.

Returns

Return type: real array

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 function `GetSoaDefinitions()`.

4.341 GetSoaResults

Returns the SOA results for the most recent simulation.

Arguments

No arguments

Returns

Return type: string

Returns an array of strings, each one describing a single SOA failure. Each string is a semi-colon delimited list with fields defined below.

Index	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

4.342 GetSymbolArcInfo

Returns information on symbol editor arc.

Arguments

No arguments

Returns

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.

4.343 GetSymbolFiles

Returns full paths of all installed symbol library files.

Arguments

No arguments

Returns

Return type: string array

4.344 GetSymbolInfo

Returns information on symbol in the symbol editor.

Arguments

No arguments

Returns

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 function Field() 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.

4.345 GetSymbolOrigin

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.

Arguments

No arguments

Returns

Return type: real array

Two element real array. Index 0 is the x-coordinate while index 1 is the y-coordinate. The units are 100 per grid square.

See Also

[SetOrigin](#)

4.346 GetSymbolPropertyInfo

Returns information about symbol editor symbol properties.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Property name

Returns

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.

If more than one property or pin is selected, the information provided in elements 0-2 will be either the property or the pin, however 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\$.

Format of result is as follows:

Index	Description
0	Property name
1	Property flags value (see Prop Attribute flags for details.)
2	Property value
3	Number of properties selected
4	Number of pins selected

4.347 GetSymbolPropertyNames

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\$”.

Arguments

No arguments

Returns

Return type: string array

4.348 GetSymbols

Returns a string array containing information about installed symbols.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	'name'	Options
2	string	No	'all'	Catalog name

Argument 1

Defines what the function returns as defined in the following table:

Options value	Description
'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 function SelectSymbolDialog() . 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.

Returns

Return type: string array

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.

4.349 GetSystemInfo

Returns information about the user's system.

Arguments

No arguments

Returns

Return type: string array

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'.
3	Available system RAM in bytes
4	Operating system class, returns 'WINNT'.
5	Operating System descriptive name.
6	Unused
7	Processor architecture
8	Operating system version (major and minor)
9	Operating service pack number
10	Number processor cores
11	Number physical processors
12	Number logical cores

4.350 GetTempFile

Creates a temporary file name

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	Default temporary directory	Directory for file

Returns

Return type: string

Returns the full path to a unique file to be used for temporary storage

Notes

The filename generated is guaranteed to be unique at the time the function executes but this function does not open the file. It is theoretically possible (but unlikely) for the filename to be used by another process between the time the function is called and at a later time when it is opened for writing.

4.351 GetTextEditorText

Returns the text of the selected text based editor. This will work for any text based editor, including the script editor and verilog editors.

Arguments

No arguments

Returns

Return type: string

Text in the currently selected text editor.

4.352 GetThreadTimes

Returns the execution times for each device thread for the most recent simulation. Requires ‘.OPTIONS devacct’ to be set for the simulation.

Arguments

No arguments

Returns

Return type: real array

Array of values of length equal to the number of threads used for the most recent simulation. Each value represents the execution time in seconds used for each device thread.

Example

4.353 GetTimerInfo

Returns information about a timer object created using [CreateTimer\(\)](#).

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Timer id

Argument 1

Timer id returned by [CreateTimer\(\)](#)

Returns

Return type:

Notes

If a timer is defined using the 'oneshot' option, the return value for the timer interval will change after the timer has triggered. Before the timer triggers the specified interval will be returned. After the timer has triggered, it will return 0.

4.354 GetTitleBlockInfo

Returns information about the selected schematic title block.

Arguments

No arguments

Returns

Return type: string array

Information about the selected title block.

Index	Description
0	Company name
1	Title
2	Author
3	Notes
4	Layout (either horizontal or vertical)
5	Logo
6	Version
7	Date

4.355 GetToolBarDefinition

Returns names of all buttons on a specified toolbar.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Toolbar name

Argument 1

Name of toolbar as defined in command [CreateToolBar](#) and populated using [DefineToolBar](#).

Returns

Return type:

String array of names of buttons that have been added to the specified toolbar using the [DefineToolBar](#) command.

See Also

[CreateToolBar](#)

[DefineToolBar](#)

[DefButton](#)

4.356 GetToolButtons

Returns name and description for available tool buttons.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	All	Button class

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](#) 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.

Returns

Return type:

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 command [DefineToolBar](#). The second value is a description of the button.

See Also

[CreateToolBar](#)

[DefButton](#)

4.357 GetTouchstoneErrors

Retrieve error messages from a [LoadTouchstone\(\)](#) call

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		ID of Touchstone data

Argument 1

ID of Touchstone data

Returns

Return type: String array

Error messages

See Also

[ReadTouchstone\(\)](#)

[DeleteTouchstone\(\)](#)

[LoadTouchstone\(\)](#)

4.358 GetUncPath

Returns the given path in UNC form. This function's main purpose is to convert windows drive letters to a consistent format.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path

Argument 1

Path of file in any form. Typically this would include a drive letter on windows.

Returns

Return type: string

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.

4.359 GetURLFromLocalPath

Encodes a local file path as a URL using the file: prefix

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		File path

Argument 1

File path in local format. This may contain a drive letter

Returns

Return type: String

4.360 GetUsedStyles

Get style names used in the current schematic

Arguments

No arguments

Returns

Return type: String array

List of style names

4.361 GetUserFile

Function opens a dialog box to allow the user to select a file.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File filter
2	string	No		Default extension
3	string	No	<<empty>>	Options
4	string	No	<<empty>>	Initial 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|*.sxsch;*.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|*.sxsch;*.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 otherwise 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 readonly 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.

Returns

Return type: string

String array of length between 1 and 3 as described in the following table:

Option 'ShowRead- Only'	Option 'Fil- terIndex'	Return value
No	No	Path name only
Yes	No	Two element array: index=0 path name index=1 Read only checked - 'TRUE' or 'FALSE'
No	Yes	Two element array: index=0 path name index=1 Filter index selected
Yes	Yes	Three element array: index=0 path name index=1 Filter index selected index=2 Read only checked - 'TRUE' or 'FALSE'

4.362 GetVecStepParameter

This function retrieves the name of the parameters that were 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.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: string

4.363 GetVecStepVals

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 there is more than one stepped parameter, the second argument may be used to identify for which parameter the values are returned.

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		Vector
2	real	No	0	index

Argument 2

Identifies parameter when there is more than one

Returns

Return type: real array

4.364 GetWidgetInfo

Returns information about open views. This is primarily an internally used function and the output may change in future releases.

Arguments

No arguments

Returns

Return type: string

Information about all the open views. In the form:

```
window_id ; tab_id ; widget_id ; widget_type ; widget_name ;
highlighted (y/n) ; window_has_focus (y/n) ;
```

4.365 GetXAxis

Returns id of selected x-axis. This function has identical behaviour to [GetSelectedXAxis\(\)](#). In version 8.1 and earlier, a graph could only have a single x-axis and this function returned its ID. For clarity we recommend that new code use the [GetSelectedXAxis\(\)](#) function.

Arguments

No arguments

Returns

Return type: string

4.366 GraphImageCapture

Opens the Graph Image Capture dialog for extracting data from a graph image. Is used in the **Digitise Data Sheet Curve** feature. The command handles initial image selection and opening.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	Current working directory	Starting directory for location of graphic files

Returns

Return type: real array

Returns the data points extracted. First element is the number of data points extracted, n . The next n elements are the x-values, the following n elements are the y-values of those data points.

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

4.367 GraphImageCaptureAdvanced

Advanced functionality of Graph Image Capture.

This provides a mechanism for setting and locking various values in the dialog. This will also provide a mechanism for obtaining the settings that were used in the dialog.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Global memory reference
2	string	No		Starting directory for location of graphic files, or empty string for not set.
3	number	No		Horizontal minimum
4	number	No		Number generated points
5	number	Yes		Horizontal maximum
6	boolean	No		Horizontal logarithmic
7	number	No		Vertical minimum
8	number	No		Vertical maximum
9	boolean	No		Vertical logarithmic

Argument 1

The reference this will write the user settings to.

Argument 3

Minimum value for the horizontal axis.

Argument 4

The number of points to generate.

Argument 5

Maximum value for the horizontal axis.

Argument 6

True if the horizontal axis is logarithmic.

Argument 7

Minimum value for the vertical axis.

Argument 8

Maximum value for the vertical axis.

Argument 9

True if the vertical axis is logarithmic.

Returns

Return type: real array

Returns the data points extracted. First element is the number of data points extracted, n . The next n elements are the x-values, the following n elements are the y-values of those data points.

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

4.368 GraphImageParameter

Returns the parameters saved to the given global location.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		The reference to return

Returns

Return type: string array

The parameters with: 0 - the image path. 1 - horizontal minimum limit. 2 - horizontal maximum limit. 3 - horizontal log flag. 4 - vertical minimum limit. 5 - vertical maximum limit. 6 - vertical log flag. 7 - number of generated points.

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

4.369 GraphLimits

Returns x and y limits of selected graph and axis type (log/linear). Function will fail if there are no selected graphs.

Arguments

No arguments

Returns

Return type: real array

The x and y axis limits of the currently selected graph and axis type. 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

4.370 GroupDelay

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		Vector

Returns

Return type: real array

4.371 Groups

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](#).

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	'name'	TitleName

Returns

Return type: string array

4.372 GuiType

Returns whether a GUI is enabled

Arguments

No arguments

Returns

Return type: string

Can return one of two values: 'none' is returned when no GUI is enabled. This is the case when the script is run from the SIM2 utility which does not have a GUI. Normally this function returns 'single' meaning that a single-window GUI style is available. (This is as opposed to earlier SIMetrix versions which used a multiple window GUI type.)

4.373 Hash

Returns a 'hash' value for the supplied string. A hash value is an integer value similar to a check sum.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Input string

Returns

Return type: string

4.374 HashAdd

Add items to a hash table. If the table is not defined as 'multiple', this function will edit the values of items that are already present. If the table is defined as 'multiple' new values for the item will be added.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Hash table id as return by HashCreate()
2	string	Yes		List of keys
3	string	Yes		List of values corresponding to the keys in argument 2

Returns

Return type: real

1.0 if hash table exists otherwise 0.0

See Also

[HashCreate\(\)](#)

[HashDelete\(\)](#)

[HashKeys\(\)](#)

[HashRemove\(\)](#)

[HashSearch\(\)](#)

4.375 HashCreate

Create a hash table.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	No	empty	Options

Argument 1

Array of strings - may be any combination of:

'temporary'

Hash table is temporary and will be automatically deleted when control returns to the command line

'multiple'

Allows multiple entries with the same name to be added to the table

Returns

Return type: real

Id of hash table. May be used in any of the hash table function. See list below in ‘See Also’ section.

Notes

Hash tables provide a fast method of searching for objects in a large list. Be aware that the number of items in the table needs to be in excess of about 10000 before the hash table offers an worthwhile improvement in performance over a linear search done using the [Search\(\)](#) function. This is because of the function overhead in the script system.

See Also

[HashDelete\(\)](#)

[HashSearch\(\)](#)

[HashAdd\(\)](#)

4.376 HashDelete

Delete a hash table

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Hash table id as return by HashCreate()

Returns

Return type: real

If hash table exists return 1.0 otherwise returns 0.0

See Also

[HashAdd\(\)](#)

[HashCreate\(\)](#)

[HashKeys\(\)](#)

[HashRemove\(\)](#)

[HashSearch\(\)](#)

4.377 HashKeys

Return a list of keys entered into a specified hash table

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Hash table id as return by HashCreate()

Returns

Return type: String array

List of keys entered into specified hash table

See Also

[HashAdd\(\)](#)

[HashCreate\(\)](#)

[HashDelete\(\)](#)

[HashRemove\(\)](#)

[HashSearch\(\)](#)

4.378 HashRemove

Remove items from a hash table

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Hash table id as return by HashCreate()
2	string	Yes		List of keys

Returns

Return type: real

Number of items deleted

See Also

[HashAdd\(\)](#)

[HashCreate\(\)](#)

[HashDelete\(\)](#)

[HashKeys\(\)](#)

[HashSearch\(\)](#)

4.379 HashSearch

Search hash table for an item

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Hash table id as return by HashCreate()
2	string array	Yes		Keys to search
3	string	No	Empty string	Empty tag

Argument 2

This can be an array provided that the table was not defined as 'multiple' on creation.

Returns

Return type: string array

For non-multiple tables, return value has the same length as argument 2. Each element maps to the corresponding element in argument 2.

For multiple tables, the return value is a list of all items that were found matching the search value.

See Also

[HashAdd\(\)](#)
[HashCreate\(\)](#)
[HashDelete\(\)](#)
[HashKeys\(\)](#)
[HashRemove\(\)](#)

4.380 HasLogSpacing

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Vector

Returns

Return type: real

4.381 HasProperty

Determines whether a particular instance possesses a specified property.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name
2	string	No		Property name to identify
3	string	No		Property value to identify
4	real	No	-1	Schematic ID

Argument 1

Property name.

Argument 2

Property name to use to identify the instance to check. If present, this argument along with argument 3, identify the instance to be tested for property ownership. If only this argument is present and not argument 3, any instance possessing the property it specifies will be tested. If neither this or argument 3 are present, the currently selected instance will be tested.

If more than instance is identified one of them will be tested but there are no rules to determine which instance will be used.

An example of this property would be 'handle'.

Argument 3

Property value to use to identify the instance to check check. If present, this argument along with argument 2, identify the instance to be tested for property ownership. If neither this or argument 3 are present, the currently selected instance will be tested.

If more than instance is identified one of them will be tested but there are no rules to determine which instance will be used.

An example of this property would be a handle name, such as 'I2'.

Argument 4

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: real

Outcome of test: TRUE (1) or FALSE (0). If no instance matches argument 2 and 3, an empty value will be returned.

4.382 HaveFeature

Determines whether a specified license feature is available.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Feature name

Argument 1

Name of license feature. Currently may be one of, 'basic', 'advanced', 'micron', 'rtn', 'simplis_if', 'AD', 'schematic' or 'scripts'.

Returns

Return type: real

Returns 1.0 if the license feature is available otherwise it returns 0.0.

4.383 HaveInternalClipboardData

Returns the number of items in the specified internal clipboard. The internal clipboard is currently only used for graph curve data.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Data type

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'.

Returns

Return type: real scalar

Notes

Use the command [CurveEditCopy](#) 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.

4.384 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.

Arguments

No arguments

Returns

Return type:

4.385 HighlightedNets

Returns names for any wholly highlighted net names on the specified schematic.

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	-1	Schematic ID

Argument 1

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string array

Returns the highlighted netnames as an array of strings.

4.386 Histogram

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Vector
2	real	Yes		Number of bins
3	string	No		Options

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.

Returns

Return type: real array

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 function [Interp\(\)](#).

4.387 HistoryDepthDialog

Displays history depth dialog box used for editing history depth for displayed curves

Arguments

Number	Type	Compulsory	Default	Description
1	String array	No	['0', '0']	Initial settings

Argument 1

Two element string array. First element is a number representing the initial option set. Valid values are '0', '1', '2' and '3'. Second element is the initial value for the Show to Depth option"

Returns

Return type: String array

Two element array in same format as argument providing the user's selection

4.388 *Iff*

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Test value
2	real array	Yes		Result if test true
3	real array	Yes		Result if test false

Returns

Return type: real array

4.389 *IffV*

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.

This function performs the same operation as [Iff\(\)](#) but also works with vectors whereas *Iff* only works with scalar values.

All three arguments may be vectors but the lengths must satisfy the following conditions:

Argument 2 (true value) must be the same length as argument 3 (false value) Argument 1 (test) must either be the same length as arguments 2 and 3 or must have a length of 1

If the test has a length greater than 1 then each element of the test is tested to select the corresponding element in the true and false vectors. If the length of the test is 1 then this value is used to select the entire vector - either the true value or false value.

The return value includes the reference value copied from argument 2. To be useful this assumes that the references of arguments 2 and 3 are the same. This would usually be the case in most applications but the function does not test this.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Test
2	real, complex, string	Yes		true value
3	real, complex, string	Yes		false value

Returns

Return type: Matches arguments 2 and 3 (must be the same)

4.390 IIR

Performs Infinite Impulse Response digital filtering on supplied vector. This function performs the operation:

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

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 function [FIR\(\)](#)) 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 function [Interp\(\)](#).

Arguments

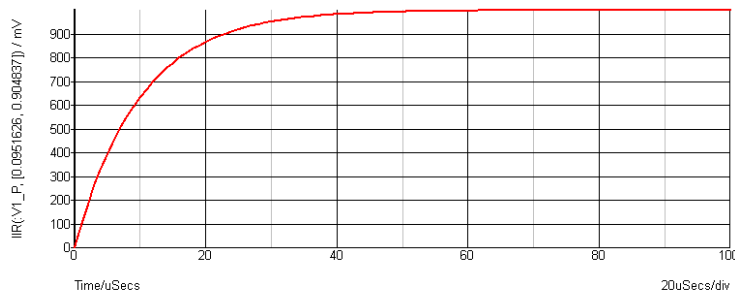
Number	Type	Compulsory	Default	Description
1	real array	Yes		Vector to be filtered
2	real array	Yes		Coefficients
3	real array	No	zero	Initial conditions

Returns

Return type: real array

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 $10 * \text{the sample interval}$. In the above the sample interval was $1\mu\text{Sec}$ so giving a $10\mu\text{Sec}$ 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:

$$y_n = x_n c_0 + y_{n-1} c_1$$

where $c_0 = 1 - \exp\left(\frac{-T}{\tau}\right)$

and $c_1 = \exp\left(\frac{-T}{\tau}\right)$

and $\tau = \text{time constant}$

and $T = \text{sample interval}$

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.

4.391 *im*

Returns imaginary part of argument, same as the function [imag\(\)](#).

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real array

Returns imaginary part of argument.

4.392 *imag*

Returns imaginary part of argument, same as the function [im\(\)](#).

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real array

Returns imaginary part of argument.

4.393 InitRandom

Initialises the random number generator used for SIMPLIS Monte Carlo distribution functions.

A seed value can be specified allowing the generator to be reset to a known state. This will allow a Monte Carlo run to be repeated to give identical results.

This function resets the random number generator used for functions [Unif\(\)](#), [Gauss\(\)](#), [GaussTrunc\(\)](#), [GaussLim\(\)](#), [Distribution\(\)](#), [UD\(\)](#) and [WC\(\)](#). These functions can only be used for evaluating expressions in a netlist processed by the pre-processor. This applies to value expressions used for components in SIMPLIS simulations.

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	seed randomly generated	seed value

Returns

Return type: real

seed used to initialise generator

Example

4.394 InputGraph

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Initial text
2	string	No	<<empty>>	Message

Returns

Return type: string

4.395 InputSchem

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Initial text
2	string	No	<<empty>>	Message

Returns

Return type: string

4.396 Instances

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 function `OpenSchematic()`. 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 function `PropValues2()`, with appropriate arguments, will return an empty *vector* when there is no match, and thus easier to use in many cases.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name
2	real	No	-1	Schematic ID

Argument 2

Schematic ID as returned by the function `OpenSchematic()`. 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.

Returns

Return type: string array

4.397 InstNets

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Options

Argument 1

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.

Returns

Return type: string array

4.398 InstNets2

Returns an array of strings holding the netnames of a schematic instance defined by arguments 1 to 3.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Schematic ID
2	string	Yes		Property name
3	string	Yes		Property value
4	string	No		Options

Argument 1

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

Argument 2

Property name to identify instance. Along with parameter 3, 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 3

Property value to identify instance. Along with parameter 2, 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.

Returns

Return type: string array

4.399 InstPins

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Property name
2	string	No		Property value
3	real	No	-1	Schematic ID

Argument 1

Property name to identify instance. Along with argument 2, 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 2

Property value to identify instance. Along with argument 1, 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 ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string array

4.400 InstPoints

Returns an array of length 3 providing XY co-ordinates and orientation of an instance.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Property name
2	string	No		Property value
3	real	No	-1	Schematic ID

Argument 1

Property name to identify instance. Along with parameter 2, 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 2

Property value to identify instance. Along with parameter 1, 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 ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: real array

Returns real array of size 3 as defined by the table. If no instance is identified by arguments 1 and 2 an empty value will be returned.

Index	Description
0	X co-ordinate
1	Y co-ordinate
2	Orientation: 0 to 7

Notes

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 the command [Inst](#) using the `/loc` switch.

4.401 InstProps

Returns an array of strings holding the names of all properties of an instance. The functions [PropValue\(\)](#) or [PropValues2\(\)](#) can be used to find values of these properties.

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	-1	Schematic ID
2	string	No		Property name
3	string	No		Property value

Argument 1

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Argument 2

Property name to identify instance. Along with parameter 2, 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

Property value to identify instance. Along with parameter 1, 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.

Returns

Return type: string array

Array of strings with property values. Returns empty value if no match to property name and value is found. Also returns empty value if the schematic ID is invalid.

4.402 integ

Integrates the argument with respect to its reference. See [Vector References](#) for details.

The function uses simple trapezoidal integration.

An error will occur if the argument supplied has no reference.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		vector

Returns

Return type: real array

4.403 *Interp*

Interpolates the data in argument 1 either to a fixed number of points or at a specified interval.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Vector to be interpolated
2	real	Yes		Number of points
3	real	No	2	Interpolation order
4	real	No		Mode

Argument 1

Vector to be interpolated. The data should have a reference (x-values, see [Vector References](#)) 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.

Returns

Return type: real array

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 function `fft()`.

4.404 IsComplex

Returns 1 (=TRUE) if the supplied argument is complex and 0 (=FALSE) if the argument is any other type.

Arguments

Number	Type	Compulsory	Default	Description
1	any	Yes		Vector

Returns

Return type: real

4.405 IsComponent

Determines whether a schematic instance is a hierarchical component. Schematic instance is defined using a property name and value.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name
2	string	Yes		Property value

Returns

Return type: real

4.406 IsDocumented

Returns whether the script command or function is documented. Also states whether the input value is a command or function.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Command or function name

Returns

Return type: integer

0 if the command or function does not exist or is not documented. 1 if it is a documented command, 2 if it is a documented function.

4.407 IsFileOfType

Returns TRUE if the filename given has a file extension contained within the set of extensions given.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Filename
2	string array	No	'global'	Extensions

Argument 1

The filename with extension to check.

Argument 2

A list of extensions to check against.

Returns

Return type: real

If the given filename has an extension contained within the extensions array, returns TRUE, otherwise returns FALSE.

4.408 IsFullPath

Returns TRUE if the supplied path name is a full absolute path.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path

Argument 1

File system path name

Returns

Return type: real

TRUE if arg is a full absolute path. FALSE if it is a relative path.

4.409 IsImageFile

SIMetrix schematics and symbols can display graphical bitmap images. This function tests whether a given image format is supported.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Image file format name

Argument 1

Image file format name including leading ‘.’. Examples include ‘.png’, ‘.jpg’ and ‘.bmp’.

Returns

Return type: string array

4.410 IsModelFile

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’.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path of file
2	string	No		Options

Returns

Return type: real

4.411 IsNum

Returns 1 (=TRUE) if the supplied argument is numeric (real or complex) and 0 (=FALSE) if the argument is a string

Arguments

Number	Type	Compulsory	Default	Description
1	any	Yes		Vector

Returns

Return type: real

4.412 IsOptionMigrateable

Determines if an option variable may be migrated in a version upgrade.

This function is used in the script that is run when SIMetrix is started for the first time. Certain option variables (defined using the command [Set](#)) 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Option name

Returns

Return type: real

Return 1.0 if the option name is migrateable otherwise returns 0.0.

4.413 IsSameFile

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. The function will always return true if the path names are identical even if the target does not exist.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path of file 1
2	string	Yes		Path of file 2

Returns

Return type: real

Returns 1 if the paths are the same, 0 otherwise.

4.414 IsScript

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Script name

Returns

Return type: real

Returns TRUE if the supplied script name can be located in the standard script path.

4.415 IsStr

Returns 1 (=TRUE) if the supplied argument is a string and 0 (=FALSE) if the argument is numeric (real or complex).

Arguments

Number	Type	Compulsory	Default	Description
1	any	Yes		Vector

Returns

Return type: real

4.416 IsTextEditor

Returns true if selected editor is a text editor. By default the selected editor will be the currently highlighted editor. Alternately argument 1 can be passed a type of editor to test for.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	<<empty>>	Text editor type

Argument 1

This can be used to search for a particular text editor type. Possible values are:

LogicDefinitionEditor

NetlistEditor

ScriptEditor

TextEditor

VerilogAEditor

VerilogHDLEditor

Returns

Return type: boolean

True or false depending on whether the selected editor is a text editor.

4.417 IsTextEditorModified

Returns true if the highlighted text editor is modified.

Arguments

No arguments

Returns

Return type: boolean

True if the highlighted editor is a text editor that has been modified, false otherwise.

4.418 Join

Joins the input string array passed as argument 1 with the separator passed as argument 2. Returns a single string with all array elements joined with the common separator.

The separator can be any string, including an empty string. Common uses for this function include joining together the ComboBox control for the [NewValueDialog\(\)](#) function, among many other uses.

This function is much faster than looping when joining large string arrays, for example a file read in by [ReadFile\(\)](#). This is due to the binary implementation of the function which is much faster than looping in a script.

This function differs significantly from `JoinStringArray()`. The `Join` function returns a single string. `JoinStringArray()` returns the concatenation of two arrays, which is a string array.

Arguments

Number	Type	Compulsory	Default	Description
1	String Array	Yes		List to Join
2	String	Yes		Separator string

Argument 1

List to Join

Argument 2

The string to join the array with. May be empty.

Returns

Return type: string

Joined string.

Example

A call to:

```
Join( [ 'a' , 'b' , 'c' ] , '|' )
```

will return the string 'abc'.

Joining on an empty string:

```
Join( [ 'a' , 'b' , 'c' ] , '' )
```

will return the string 'abc'.

Joining a string array of one element:

```
Join( [ 'a' ] , ',' )
```

will return the string 'a'. In this case the separator is ignored.

Reading in a file as a single string:

```
Let file_array = ReadFile( 'data.txt' )
Let file_as_string = Join( file_array , '' )
```

The resulting variable `file_as_string` will be a single string containing all the text of the `data.txt` file.

See Also

- [JoinStringArray\(\)](#)
- [Parse\(\)](#)
- [Scan\(\)](#)
- [ParseEscape\(\)](#)
- [ScanEscape\(\)](#)

4.419 JoinStringArray

Concatenates two string arrays to return a single array. Either argument may be a zero-length vector

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		First array
2	string array	Yes		Second array

Returns

Return type: string array

Array of strings of length equal to the sum of the lengths of arguments 1 and 2. Contains arguments 1 and 2 concatenated together.

See Also

- [JoinVectors\(\)](#)
- [MakeString\(\)](#)

4.420 JoinVectors

Concatenates two vectors of the same type. Works with vectors of type string, real or complex. Either argument may be a zero-length vector

Arguments

Number	Type	Compulsory	Default	Description
1	Real, complex or string. Must be the same type as arg 2	Yes		First vector
2	Real, complex or string. Must be the same type as arg 1	Yes		Second vector

Argument 1

First vector

Argument 2

Second vector

Returns

Return type: Same as arguments

Concatenation of the two arguments

See Also

[JoinStringArray\(\)](#)

[MakeString\(\)](#)

4.421 *length*

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.

Arguments

Number	Type	Compulsory	Default	Description
1	any	Yes		Vector

Returns

Return type: real

4.422 *ListDirectory*

Lists all files that comply with the spec provided in argument 1.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path specification
2	string	No	'none'	Option

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.

Returns

Return type: string array

4.423 ListSchemProps

Returns the schematic properties.

Arguments

No arguments

Returns

Return type: string array

The schematic property names and whether they are writeable or readonly.

4.424 ln

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Returns the natural logarithm of the argument.

4.425 LoadFile

Returns an array of strings holding lines of text from the file specified by argument 1.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File name
2	string	No	'auto'	Encoding

Argument 2

Character encoding assumed for input file. May be any value returned by the function [GetCodecNames\(\)](#). Examples include:

'utf-8' UTF8 encoding. This is the encoding used internally and for output

'utf-16' UTF16 also known as UCS-2

'Shift-JIS' Commonly used on Japanese systems

In addition the following special values may be used:

'locale' uses the default encoding for the system's locale

'auto' uses 'utf-8' if successful. Otherwise uses 'locale'

Returns

Return type: string array

4.426 LoadSensitivityReport

Each sensitivity analysis generates an XML file containing the results of that analysis. This includes the evaluation of the goal functions and and the component values used for each case. This function displays that data in a table view

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Input file

Argument 1

Sensitivity XML file path

Returns

Return type: real

Status: success: 1.0, fail: 0.0

4.427 LoadTouchstone

Loads a Touchstone network parameter file and converts it to Y-parameter values.

LoadTouchstone returns an ID for subsequent access to the data in the file or error messages if the load failed.

If the load was successful, use the function [ReadTouchstone\(\)](#) to read its data. If unsuccessful, error messages may be read using [GetTouchstoneErrors\(\)](#).

Use [DeleteTouchstone\(\)](#) to delete the data loaded by LoadTouchstone.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Filename
2	Real	Yes		Order of network

Argument 1

Path to Touchstone file

Argument 2

Order of network, that is the number of ports

Returns

Return type: Real array

Two field real array. First field is an integer ID than can be used for functions [ReadTouchstone\(\)](#).. Second field is 1 if the file was successfully read, otherwise 0.

Notes

Currently only 1 and 2 port s-parameter data and y-parameter data of any size is accepted. The data is returned as y-parameter values.

See Also

[ReadTouchstone\(\)](#)

[GetTouchstoneErrors\(\)](#)

[DeleteTouchstone\(\)](#)

4.428 Locate

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](#)) 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.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Vector
2	real	Yes		Search value

Returns

Return type: real

4.429 log

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.

This is identical to [log10\(\)](#). 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.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Returns log to base 10 of argument.

4.430 log10

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Returns log to base 10 of argument.

4.431 LPF

Applies a continuous time single pole low pass filter to input data. Unlike the [IIR\(\)](#) and [FIR\(\)](#) functions, this function does not require the input data to be sampled. That is it can use simulation data directly without requiring interpolation.

This function can be used with multi-division data.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Input data
2	Real	Yes		Time constant
3	Real	No	First input value	Initial condition

Argument 1

Input data. This is expected to be an XY vector with both X and Y values. The function will return an empty vector if this is not the case.

Argument 2

Time constant

Argument 3

Initial condition. Specifies the output value for the first point

Returns

Return type: real array

Result

4.432 mag

Returns the magnitude of the argument. This function is identical to the [abs\(\)](#) function.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real array

Magnitude of the input argument

4.433 magnitude

Returns the magnitude of the argument. This function is identical to the [mag\(\)](#) function.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real array

Magnitude of the input argument

4.434 makecomplexvec

Returns a complex vector consisting of all 0's. Argument specifies length of vector.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		

Argument 1

Number of elements in result

Returns

Return type: real array

4.435 MakeDir

Creates the directory specified by arg 1. Returns 0 if successful otherwise returns 1.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Directory name

Returns

Return type: real

4.436 MakeLogicalPath

Converts a file system path to a symbolic path using the automatic path matching mechanism. This process is described in *User's Manual/Sundry Topics/Symbolic Path Names*.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path
2	string	Yes		Options

Argument 2

May be set to either one of:

Name	Description
'systemonly'	Will only match system symbols to the path. These are: %STARTPATH% %DOCSPATH% %EXEPATH% %APPDATA% %TEMP% %TEMPPATH% %SXAPPDATA% %SHAREPATH% %LIBPATH% %SXDOCSPATH% %COMMON_APPDATA%
'projectonly'	Will only match symbols listed in the [Locations] section of the configuration file

Refer to *User's Manual/Sundry Topics/Symbolic Path Names/Definition* for details of system path

Returns

Return type: string

4.437 MakeString

Creates an array of strings. Length of array is given as argument to function. The strings may be initialised by supplying argument 2.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Number of elements in result
2	string array	No		Initial values

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`.

Returns

Return type: string array

Returns new string

4.438 makevec

Returns a vector consisting of all 0's. Scalar argument specifies length of vector. If argument is itself a vector, the return value is a multi-division vector whose division lengths are defined by the value sin each field

Arguments

Number	Type	Compulsory	Default	Description
1	real scalar or array	Yes		Number of elements in result or each division in result

Argument 1

Number of elements in result if argument is a scalar. If the argument is a vector, the return value will be a multi-division vector with a number of divisions equal to the length of this argument. The length of each division will be equal to the corresponding entry in this argument

Returns

Return type: real array

Vector or multi-division vector with all values set to zero

4.439 ManageDataGroupsDialog

Specialised function that opens the Manage Data Group dialog box. The box displays data group information in tabular form with each row representing a single group. The box allows editing of the information and also for groups to be deleted.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		List of data groups and associated information

Argument 1

String array with each element comprising a semi-colon delimited list of items that describe a single group. The items are as follows:

Field	Description
0	Group name
1	Group title
2	Analysis mode
3	Flags: a combination of 'current', 'global', 'keep'

Returns

Return type: string array

String array of the same length as argument 1. Each array element comprising a semi-colon delimited list of items as follows:

Field	Description
0	Group name
1	Group title

Field	Description
2	Flags: a combination of 'current', 'global', 'keep' and 'delete'

Items marked 'delete' were deleted by the user.

The function will return an empty vector if the Cancel button is clicked.

4.440 ManageMeasureDialog

Opens dialog box used to manage graph measurements.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		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:

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

Returns

Return type: string array

4.441 MapArray

Creates an array of real or string items listed in argument 1 with array locations defined in argument 2.

Arguments

Number	Type	Compulsory	Default	Description
1	real or string	Yes		Items to be included in result
2	real	Yes		Indexes for corresponding items in argument 1

Argument 1

Items to be included in result

Argument 2

Indexes for corresponding items in argument 1

Returns

Return type: Same as argument 1

Array of the same type as argument 1 with length equal to the highest value found in argument 2.

The return value will be an array where each element is an item in argument 1 located at the index specified in the corresponding location in argument 2.

Unused locations will contain empty strings if argument 1 is of type string or -1 if argument 1 is of type real.

4.442 max

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		vector 1
2	real	Yes		vector 2

Returns

Return type: real array

4.443 maxidx

Returns index of the input array element with largest magnitude.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Returns index to maximum input value

4.444 Maxima

Returns array of values holding every maximum point in the supplied vector whose value complies with limits specified in argument 2.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Vector
2	real array	No	$[-\infty, +\infty]$	[min limit, max limit]
3	string	No	$\langle\langle\text{empty}\rangle\rangle$	Options
4	real	No	0.0	Tolerance

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. Values are:

- 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 not 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.

Argument 4

Minimum spacing between x values. Any pair of points that are closer than this value will be treated as a single point

Returns

Return type: real array

The function returns the XY values for each maximum point. The X-values are returned as the vector’s reference (see [Vector References](#)).

4.445 Maximum

Returns the largest value found in the vector specified in argument 1 in the range of x values specified by arguments 2 and 3.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		Vector
2	real	No	start of vector	Min range
3	real	No	end of vector	Max range

Returns

Return type: Real

4.446 mean

Returns the average of all values in supplied argument. If the argument is complex the result will also be complex.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Returns the average of the supplied arguments

4.447 Mean1

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Input vector
2	real	No	Start of input vector	start x value
3	real	No	End of input vector	end x value

Returns

Return type: real

4.448 MeasureDialog

Opens dialog for specifying graph measurements.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	No		Dialog data
2	string array	No		Initial settings
3	string array	No		Condition

Argument 1

Dialog data. Format the same as for argument 1 in the function [ManageMeasureDialog\(\)](#) 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.

Returns

Return type: string array

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

4.449 MessageBox

Opens a message dialog box with a choice of styles.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Message
2	string array	No		Options

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 Classic'

Argument 2

1, 2 or 3 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'

Third element may be 'dontaskagain' or 'dontshowmessageagain' in which case a 'Do not ask again' or 'Do not show this message again' check box will also be displayed

Returns

Return type: string

A single string indicating the user's response. One of:

'Abort'

'Cancel'

'Ignore'

'No'

'Ok'

'Retry'

'Yes'

If 'dontaskagain' or 'dontshowmessageagain' was specified and the user checked the 'Do not ask again'/'Do not show this message again' check box, the return value will be appended with 'IDontAskAgain'

4.450 Mid

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		String
2	real	Yes		Start index
3	real	No	to end of string	Length of result

Returns

Return type: string

Example

```
Mid('Hello World!', 6, 5)
```

will return 'World'.

See Also

[Char\(\)](#)

4.451 min

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Vector 1
2	real	Yes		Vector 2

Returns

Return type: real array

4.452 minidx

Returns index of the input array element with smallest magnitude.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Returns index to minimum input value

4.453 Minima

Returns array of values holding every minimum point in the supplied vector whose value complies with limits specified in argument 2.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Vector
2	real array	No	$[-\infty, +\infty]$	[min limit, max limit]
3	string	No	$\langle\langle\text{empty}\rangle\rangle$	Options
4	real	No	0.0	Tolerance

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. Values are:

0	Maximum limit i.e. minima must be above this to be accepted
1	Minimum limit i.e. minima 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 not 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.

Argument 4

Minimum spacing between x values. Any pair of points that are closer than this value will be treated as a single point

Returns

Return type: real array

The function returns the XY values for each minimum point. The X-values are returned as the vector's reference (see [Vector References](#)).

4.454 Minimum

Returns the smallest value found in the vector specified in argument 1 in the range of x values specified by arguments 2 and 3.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		Vector
2	real	No	start of vector	Min range
3	real	No	end of vector	Max range

Returns

Return type: Real

4.455 MkVec

Most simulation vectors are accessed using the name of the node that generated the data. For example if a node is called 'VOUT' the vector to access the data on that node is also called 'VOUT'.

However, some nodes are named in a manner that cannot directly be accessed as the name contains characters that can be confused with arithmetic and other operators. For example, it is legal to call a node +15V but this would be confused with the constant value +15.

To resolve this, a vector may be accessed using the [Vec\(\)](#) function. E.g. `Vec('+15V')`. The `MkVec()` function will return a string that can be used to access the vector data. If the vector name does not contain any conflicting characters, it will return the name unmodified. If it does contain conflicting characters, it will return a string using the `Vec` function.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Node name

Returns

Return type: string

Expression to access node data

4.456 MLRidgeRegressionFit

Performs a ridge regression based line fit to a set of data, producing a polynomial curve that fits the data.

Given a set of training parameters and observations (x and y values) along with a parameter controlling the smoothness of the required output, the function returns a set of values that make up a curve that fits to the parameters and observations.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Training parameters (x values)
2	real array	Yes		Training observations (y values)
3	real array	Yes		Polynomial
4	real array	Yes		Regularisation parameter
5	real array	Yes		Result parameters (x values)

Argument 1

The parameters for the training data. This would normally be the values on the x-axis of a graph. The values must be ordered from lowest to highest value.

Argument 2

The observations for the training data. This would normally be the values on the y-axis of a graph.

Argument 3

The degree of polynomial to fit the data to.

Argument 4

Parameter that controls how smooth the fit to the data will be. Value must be 0-positive, where the smoothness of the fit increases as the parameter increases.

At the extremes, a value of 0 produces a resulting that will try to pass through all given data points, subject to the flexibility within the polynomial chosen, whilst a value tending towards infinity produces a single straight line through the whole of the data.

Argument 5

The parameters to fit the resulting curve to.

Returns

Return type: real array

Vector the same length as parameter 5 (*Result parameters (x values)*), with fitted values for each parameter in order.

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

4.457 MLSplineFit

Performs a spline based line fit to a set of data.

Given a set of training parameters and observations (x and y values) along with a parameter controlling the smoothness of the required output, the function returns a set of values that make up a curve that fits to the parameters and observations.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Training parameters (x values)
2	real array	Yes		Training observations (y values)
3	real array	Yes		Smoothness parameter
4	real array	Yes		Result parameters (x values)

Argument 1

The parameters for the training data. This would normally be the values on the x-axis of a graph. The values must be ordered from lowest to highest value.

Argument 2

The observations for the training data. This would normally be the values on the y-axis of a graph.

Argument 3

Parameter that controls how smooth the fit to the data will be. Value must be 0-positive, where the smoothness of the fit increases as the parameter increases.

At the extremes, a value of 0 produces a result made up of straight lines between each training point in order, whilst a value tending towards infinity produces a single straight line through the whole of the data.

Argument 4

The parameters to fit the resulting curve to.

Returns

Return type: real array

Vector the same length as parameter 4 (*Result parameters (x values)*), with fitted values for each parameter in order.

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

4.458 MLVector

Creates a vector of consecutively increasing values from a minimum to maximum value using a given increment.

Eg. `MLVector(0,1,10)` would give: `[0 1 2 3 4 5 6 7 8 9 10]`.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Minimum value
2	real	Yes		Increment value
3	real	Yes		Maximum value

Returns

Return type: real array

A vector of values within the specified range with given increments.

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

4.459 ModelLibsChanged

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.

Arguments

No arguments

Returns

Return type: real

4.460 MSWReadHeader

Read header information of an MSW file. The header contains the analysis line used to create it along with the simulation group

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		File to read

Argument 1

File to read

Returns

Return type: String array

Header information in order:

analysis-line, groupname, date, title.

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

4.461 Navigate

Returns path name of hierarchical block given root path and full component reference.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Full component reference
2	string	Yes		Path of hierarchical root

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.

Returns

Return type: string

Returns path name of schematic hierarchical block.

4.462 NearestInst

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name

Returns

Return type: string

See Also

[Branch\(\)](#)

[NetName\(\)](#)

[PinName\(\)](#)

4.463 NetName

Returns the net name of the nearest wire or instance pin.

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	<<empty>>	Option

Argument 1

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).

Returns

Return type: string

Returns the net name of the nearest wire or instance pin.

4.464 NetNames

Returns array of all net names in selected schematic

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	-1	Schematic ID

Argument 1

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string array

Returns an array of strings holding all the net names in the currently selected schematic. Returns an empty value if the schematic is empty or can't be found.

4.465 NetWires

Returns wire handles of names net.

Note that this function requires that the schematic has been netlisted. This can be forced using the function [Netlist](#) in the form:

```
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')

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Net name
2	real	No	-1	Schematic ID

Argument 1

Name of net whose wire handles are required.

Argument 2

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

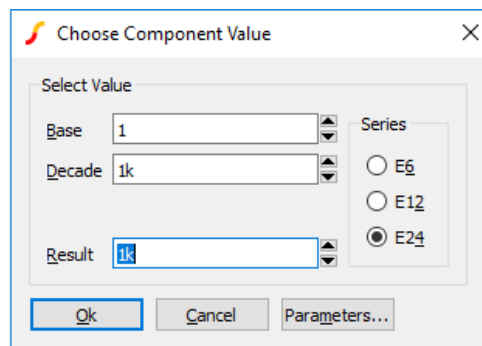
Return type: string array

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.

4.466 NewPassiveDialog

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)
```



Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Initial value
2	string array	No	['Select value', 'E12']	[message series]
3	string array	No	<<empty>>	Parameter names
4	string array	No	<<empty>>	Parameter values

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.

Returns

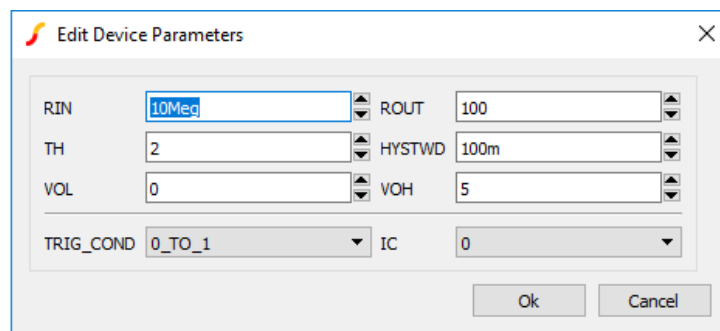
Return type: string array

The function returns a string array in the following form:

Index	Description
0	Value in “Result” box
1	Number of parameter values
2	The values of the parameters in the order they were passed (onwards values)

4.467 NewValueDialog

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:



Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Control definitions
2	string	Yes		Initial values
3	string	Yes		Options

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` is a text string defining the control's label, which may not contain the characters ':' or '|'.

`type` is one of the following:

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.

Returns

Return type: string array

Example

The following call would display the dialog as shown above.

```
Show 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'])
```

4.468 *norm*

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Returns the norm of the input.

4.469 NumberSelectedAnnotations

Returns the number of selected annotations.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	empty string	Filter

Argument 1

Optional filter string. If set to “*textenabled*” only annotations that text can be added to are counted.

Returns

Return type: real

The number of selected annotations.

4.470 NumDivisions

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](#).

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		Vector

Returns

Return type: real

4.471 NumElems

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.

Arguments

Number	Type	Compulsory	Default	Description
1	any	Yes		Vector

Argument 1

Vector. A zero-length vector is permitted

Returns

Return type: real array

4.472 OpenEchoFile

Redirects the output of the command [Echo](#) to a file. Redirection is disabled when the function [CloseEchoFile\(\)](#) is called or when control returns to the command line.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File name
2	string	Yes		Access mode

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.

Returns

Return type: real

4.473 OpenFile

Opens a file and returns its handle. This may be used by the command [Echo](#). Use the function [CloseFile\(\)](#) to close the file.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File path
2	string	Yes		File open mode

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.

Returns

Return type: real

4.474 OpenPDFPrinter

Sets up printing for PDF output.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Filename

Returns

Return type: String

Success or Failed message.

4.475 OpenPrinter

Starts a print session. This may be used for customised or non-interactive printing. See [Non-interactive and Customised Printing](#)

Arguments

Number	Type	Compulsory	Default	Description
1	string array	No		Configuration

Argument 1

String array with up to 6 elements as described in the following table

Index	Description
0	Print orientation: 'landscape' or 'portrait'
1	Number of copies
2	Printer id. This is an index and can be found from the function GetPrinterInfo() . If omitted, the application default printer will be used.
3	Title of print job. 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. title is not printed on the final document.
4	Specify printer by name. If omitted, printer will be defined by its index (see above) or the application default printer will be used

Returns

Return type: string

Status of operation: either 'Success' or 'Failed'

4.476 OpenSchem

Opens a schematic similar to the command [OpenSchem](#) but returns a code indicating success or otherwise.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File path
2	string array	No		Options

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

Returns

Return type: string

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

4.477 OpenSchematic

Opens a schematic given its file system path. The return value may be used with a number of other functions and commands. This function does not display the schematic.

The function [GetSchematicTabs\(\)](#) returns the IDs for all currently displayed schematics.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File path

Argument 1

File system path to schematic file. The schematic does not need to be currently displayed

Returns

Return type: real

Returns an integer ID that can be used for a wide range of functions that return information about a schematic. It may also be used by some commands. 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 IDs, allow information to be retrieved from schematics that are not currently on display. If the specified schematic is displayed then the values returned by the supported functions will reflect the state of the displayed schematic and not the saved schematic.

The return value from `OpenSchematic` can be used with the following functions:

- `CloseSchematicTab()`
- `DescendHierarchy()`
- `ElementProps()`
- `GetChildModulePorts()`
- `GetComponentValue()`
- `GetFullLines()`
- `GetInstancePinLocs()`
- `GetModifiedStatus()`
- `GetReadOnlyStatus()`
- `GetSchematicTabs()`
- `GetSchematicVersion()`
- `GetSimulatorMode()`
- `HasProperty()`
- `HighlightedNets()`
- `Instances()`
- `InstNets2()`
- `InstPins()`
- `InstPoints()`
- `InstProps()`
- `NetNames()`
- `NetWires()`
- `PropFlags()`
- `PropFlags2()`
- `PropFlagsAll()`
- `PropFlagsAnnotations()`
- `PropFlagsWires()`
- `PropValues()`
- `PropValues2()`
- `SetComponentValue()`

[SetReadOnlyStatus\(\)](#)
[SymbolName\(\)](#)
[SymbolNames\(\)](#)
[WirePoints\(\)](#)
[Wires\(\)](#)

The schematic ID may also be used by these commands:

[SaveAs](#)
[SelectSchematic](#)

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.

4.478 OptimiserAnalysisLine

Returns optimiser analysis lines

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)
[OptimiserCreateFromXML\(\)](#)
[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: String array

Returns an array with each element containing one optimiser analysis line. The final element holds any options line present in the definition. If there are no options lines, the final element will be an empty string

4.479 OptimiserApplyFailedRun

Notify optimiser that a simulation run failed. Currently this will abort the optimisation process.

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID
2	String	No	Nomessage	Error message

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)

[OptimiserCreateFromXML\(\)](#)

[OptimiserWidgetCreateOptDef\(\)](#)

[OptimiserSimulatorGetDef\(\)](#)

Argument 2

Error message which will be recorded in final report

Returns

Return type: Real array

2 element array with both values set to zero

4.480 OptimiserApplySpecification

Evaluates measurements for specified optimiser definition and passes results to the optimiser. Assumes that optimiser session was started with [OptimiserStart\(\)](#) and also assumes that all analysis IDs were specified on analysis lines. This is automatically handled for SIMPLIS simulations as long as the optimiser ID is passed to the PreProcessNetlist command using the /optid switch

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)

[OptimiserCreateFromXML\(\)](#)

[OptimiserWidgetCreateOptDef\(\)](#)

[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: Real array

Real array

0	number of measurements (0 if measurements failed)
1	1: measurements Ok, 0: measurements failed
next num_measurements :	1:measurement OK, 0:measurement failed
next num_measurements :	Measurement values

Returns empty if ID invalid or if optimiser is not running

See Also

- [OptimiserFinish\(\)](#)
- [OptimiserRunning\(\)](#)
- [OptimiserApplyFailedRun\(\)](#)
- [OptimiserSuccess\(\)](#)
- [OptimiserErrorMessage\(\)](#)
- [OptimiserStart\(\)](#)

4.481 OptimiserCloseDef

Deletes an optimiser object defined by its ID. Note that this deletes the ID and does not necessarily delete the object that created it. For example, [OptimiserSimulatorGetDef\(\)](#) returns an ID providing details of the current state of the simulator's optimiser. Calling [OptimiserCloseDef](#) on that ID will not destroy the simulator's optimiser it only destroys the data that was generated from that optimiser.

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

- [OptimiserCreateFromXMLString\(\)](#)
- [OptimiserCreateFromXML\(\)](#)
- [OptimiserWidgetCreateOptDef\(\)](#)
- [OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: Real

Returns 1 if object existed and was successfully deleted otherwise returns 0. A return value of 0 usually means that the ID was not valid.

4.482 OptimiserCreateFromXML

Create an optimiser definition from an XML file. The optimiser definition including its parameters, analyses and measurements can be described using an XML based definition. This function loads such a definition from a file and returns a reference ID.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Filename of XML definition file

Argument 1

Filename of XML definition file

Returns

Return type: Real

ID of optimiser definition, -1 on failure

4.483 OptimiserCreateFromXMLString

Creates an optimiser definition from an XML string

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Optimiser XML definition

Argument 1

Optimiser XML definition. The optimiser definition including its parameters, analyses and measurements can be described using an XML based definition. The following functions return an XML definition in string form:

[OptimiserSimulatorWriteXMLString\(\)](#)

[OptimiserWriteXMLString\(\)](#)

Returns

Return type: Real

ID of optimiser definition

4.484 OptimiserErrorMessage

Returns string showing error message returned by a filed optimiser run

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)

[OptimiserCreateFromXML\(\)](#)

[OptimiserWidgetCreateOptDef\(\)](#)

[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: String

Optimiser error message. Empty if optimiser definition ID is invalid.

4.485 OptimiserFinish

Instruct optimiser runtime session to terminate. This function can only be used to stop a session started with OptimiserStart typically used for SIMPLIS simulations

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)

[OptimiserCreateFromXML\(\)](#)

[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: Real

Status. Returns 1 if optimiser was previously running otherwise returns 0. Returns -1 if definition invalid

See Also

[OptimiserRunning\(\)](#)
[OptimiserApplySpecification\(\)](#)
[OptimiserApplyFailedRun\(\)](#)
[OptimiserSuccess\(\)](#)
[OptimiserErrorMessage\(\)](#)
[OptimiserStart\(\)](#)

4.486 OptimiserGetDataObject

Get data for optimiser data object given optimiser definition ID nad data object name

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID
2	String	Yes		Name of data object

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)
[OptimiserCreateFromXML\(\)](#)
[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Argument 2

Name of data object

Returns

Return type: Real vector

XY Vector containing both x-values and y-values for data object

See Also

[OptimiserGetDataObjectNames\(\)](#)

4.487 OptimiserGetDataObjectNames

Returns names of all data objects in an optimiser definition. Name can be used as an argument to [OptimiserGetDataObject\(\)](#) to get actual data. Note that data objects are not stored in results files nor in the optimiser definition returned by the simulator (using [OptimiserSimulatorWriteXMLString\(\)](#))

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)

[OptimiserCreateFromXML\(\)](#)

[OptimiserWidgetCreateOptDef\(\)](#)

[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: String array

List of names. Returns an empty vector if ID invalid

See Also

[OptimiserGetDataObject\(\)](#)

4.488 OptimiserGetInfo

Get optimiser file information

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)

[OptimiserCreateFromXML\(\)](#)

[OptimiserWidgetCreateOptDef\(\)](#)

[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: String array

- 0 Path to attached schenmatic
- 1 Path to results file
- 2 Path to definition file

Returns an empty vector if ID invalid

See Also

[OptimiserSimulatorGetInfo\(\)](#)

[OptimiserWidgetGetInfo\(\)](#)

4.489 OptimiserGetIteration

Returns data from a single optimiser iteration

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID
2	Real	Yes		Iteration index

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)

[OptimiserCreateFromXML\(\)](#)

[OptimiserWidgetCreateOptDef\(\)](#)

[OptimiserSimulatorGetDef\(\)](#)

Argument 2

Iteration index

Returns

Return type:

Real array with results in three groups:

First group: parameter values

Second group: measurement values

Third group: Constraint pass/fail for each measurement. 1.0 for pass 0.0 for fail. Includes objective which is always 1.0

4.490 OptimiserGetOptions

Returns options for an optimiser definition

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)

[OptimiserCreateFromXML\(\)](#)

[OptimiserWidgetCreateOptDef\(\)](#)

[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: String array

5 element string array:

- 0 Iteration limit. 0 if not defined
- 1 relative tolerance
- 2 absolute tolerance
- 3 show progress (1 enabled, 0 not enabled)

Returns empty if ID invalid

4.491 **OptimiserGetParameters**

Returns parameter definition from an optimiser definition

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)
[OptimiserCreateFromXML\(\)](#)
[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: String array

String array with one element for each parameter. Each element is a semi-colon delimited string with four fields:

- 0 Parameter name
- 1 Initialvalue
- 2 Minimum value or empty if no minimum defined
- 3 Maximum value or empty if no maximum defined

Returns an empty vector if ID invalid

See Also

[OptimiserGetSpecification\(\)](#)

4.492 **OptimiserGetParameterValues**

Gets parameter values for current optimiser iteration

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)
[OptimiserCreateFromXML\(\)](#)
[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: Real array

Array of parameter values for current iteration in same order as [OptimiserGetParameters\(\)](#).

Returns empty if the ID is invalid or if the optimiser is not running.

4.493 OptimiserGetSpecification

Returns details of all analyses and specifications for an optimiser definition

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)
[OptimiserCreateFromXML\(\)](#)
[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type:

Array of strings providing analysis definitions along with all measurements. Definition as follows:

0	Number of analyses
1	Number of specifications
Next num_analyses elements	semi-colon delimited string in form analysis_id;analysis_line
Next num_measurements elements	semi-colon delimited string in form analysis_id ; label ; expression ; value ; tolerance

4.494 OptimiserLoadWidgetFromXML

Loads an optimiser widget (GUI) from an XML file

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Filename

Argument 1

Path to file to load into widget

Returns

Return type:

Returns 1 if successful, 0 if file failed to load, -1 if there is no current valid optimiser widget

4.495 OptimiserParameterLine

Gets parameter line for current optimiser iteration

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)
[OptimiserCreateFromXML\(\)](#)
[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: String

Semicolon delimited string. Each element in form name=value giving the current value for each optimiser parameter.

Returns empty if the ID is invalid or if the optimiser is not running.

4.496 OptimiserResults

Returns information about the results of an optimiser run

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)

[OptimiserCreateFromXML\(\)](#)

[OptimiserWidgetCreateOptDef\(\)](#)

[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: Real array

Real array

0	Order (i.e. number of parameters)
1	Number of measurements
2	Iteration number of best result
3	Value of objective function for best result
4	1: complete, 0: not complete
5	number of iterations
next order	parameter values
next measurement count	measurement values

4.497 OptimiserRunning

Returns 1 when optimiser run is currently running. Returns 0 if complete or not started

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)
[OptimiserCreateFromXML\(\)](#)
[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type:

Returns 1 if optimiser is running, 0 if is complete or not started. Returns -1 if ID is invalid

See Also

[OptimiserFinish\(\)](#)
[OptimiserApplySpecification\(\)](#)
[OptimiserApplyFailedRun\(\)](#)
[OptimiserSuccess\(\)](#)
[OptimiserErrorMessage\(\)](#)
[OptimiserStart\(\)](#)

4.498 OptimiserSimulatorGetDef

Returns an optimiser ID for the most recent simulator's optimiser run. The returned ID can be used for any optimiser function. Note that the ID captures the current state of the simulator's optimiser definition

Arguments

No arguments

Returns

Return type: Real

Optimiser ID

4.499 OptimiserSimulatorGetInfo

Get simulator optimiser file information

Arguments

No arguments

Returns

Return type: String array

- 0 Path to attached schematic
- 1 Path to results file
- 2 Path to definition file

Returns an empty vector if no simulator optimiser is current

See Also

[OptimiserGetInfo\(\)](#)

[OptimiserWidgetGetInfo\(\)](#)

4.500 OptimiserSimulatorResults

Returns information about the results of a simulator optimiser run.

Arguments

No arguments

Returns

Return type: Real array

Real array

- | | |
|------------------------|---|
| 0 | Order (i.e. number of parameters) |
| 1 | Number of measurements |
| 2 | Iteration number of best result |
| 3 | Value of objective function for best result |
| 4 | 1: complete, 0: not complete |
| 5 | number of iterations |
| next order | parameter values |
| next measurement count | measurement values |

4.501 OptimiserSimulatorStatus

Returns current status of the simulator's optimiser as a string

Arguments

No arguments

Returns

Return type: String array

Two element array. First element:

success	Optimiser has completed successfully
notstarted	Optimiser has not been started
running	Optimiser is running
failed	Optimiser failed
unknown	Unknown status

Second element returns one of the following codes which provides more detail:

not_started	Optimiser not started
running	Optimiser running
success	Optimiser completed successfully. This is not usually returned, instead one of the reasons for the optimisation failure is returned.
stopvalreached	Optimiser objective function reached its stop value
ftolreached	Optimiser objective function converged
xtolreached	[Future use] Optimiser parameter values converged
maxitersreached	Optimiser iteration count reached upper limit
maxtimereached	[Future use] Optimiser execution time reached upper limit
failed	Optimiser failed
invalidargs	[Internal error] Arguments to optimiser invalid
outofmemory	Optimiser out of memory
roundofflimited	Optimiser reached accuracy limit
forcedstop	Optimiser forced stop
convergencefail	Simulator convergence failed
simulationfailed	Other simulator failure
measurementfailed	Measurement evaluation failed
userrequest	User requested optimiser analysis abort
unknown	Unknown error
invalid	No simulator optimiser available

See Also

[OptimiserStatus\(\)](#)

4.502 OptimiserSimulatorUserMessage

Returns a string providing the current state of the simulator's optimiser for display to the user. The message includes the current iteration number, parameter values and resulting measurements

Arguments

No arguments

Returns

Return type:

String giving current iteration number, parameter values and resulting measurements. Empty if there is no current simulator optimiser

See Also

OptimiserUserMessage

4.503 OptimiserSimulatorWriteXMLString

Returns the XML definition of the current state of the simulator's optimiser as a string.

Arguments

No arguments

Returns

Return type: String

XML definition as a string. Empty if there is no current simulator optimiser

See Also

[OptimiserWriteXMLString\(\)](#)

4.504 OptimiserStart

Start optimiser run. This function is used for running optimisation for the SIMPLIS simulator. This function creates the runtime environment needed to run an optimisation session.

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)
[OptimiserCreateFromXML\(\)](#)
[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: Real

Status. Rreturn 1 if successful, 0 on failure

See Also

[OptimiserRunning\(\)](#)
[OptimiserApplySpecification\(\)](#)
[OptimiserApplyFailedRun\(\)](#)
[OptimiserSuccess\(\)](#)
[OptimiserErrorMessage\(\)](#)
[OptimiserFinish\(\)](#)

4.505 OptimiserStatus

Returns current status of an optimiser as a string

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)
[OptimiserCreateFromXML\(\)](#)
[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type:

Two element array. First element:

success	Optimiser has completed successfully
notstarted	Optimiser has not been started
running	Optimiser is running
failed	Optimiser failed
unknown	Unknown status

Second element returns one of the following codes which provides more detail:

not_started	Optimiser not started
running	Optimiser running
success	Optimiser completed successfully. This is not usually returned, instead one of the reasons for the optimis
stopvalreached	Optimiser objective function reached its stop value
ftolreached	Optimiser objective function converged
xtolreached	[Future use] Optimiser parameter values converged
maxitersreached	Optimiser iteration count reached upper limit
maxtimereached	[Future use] Optimiser execution time reached upper limit
failed	Optimiser failed
invalidargs	[Internal error] Arguments to optimiser invalid
outofmemory	Optimiser out of memory
roundofflimited	Optimiser reached accuracy limit
forcedstop	Optimiser forced stop
convergencefail	Simulator convergence failed
simulationfailed	Other simulator failure
measurementfailed	Measurement evaluation failed
userrequest	User requested optimiser analysis abort
unknown	Unknown error
invalid	ID invalid

See Also

[OptimiserSimulatorStatus\(\)](#)

4.506 OptimiserSuccess

Returns 1 if optimiser run has completed successfully

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)

[OptimiserCreateFromXML\(\)](#)

[OptimiserWidgetCreateOptDef\(\)](#)

[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: Real

Returns 1 if optimiser run has completed successfully, 0 if unsuccessful and -1 if ID not recognised

4.507 OptimiserUserMessage

Returns a string providing the current state of a running optimiser for display to the user. The message includes the current iteration number, parameter values and resulting measurements

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)

[OptimiserCreateFromXML\(\)](#)

[OptimiserWidgetCreateOptDef\(\)](#)

[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: String

String giving current iteration number, parameter values and resulting measurements. Empty if optimiser definition ID is invalid

See Also

[OptimiserSimulatorUserMessage](#)

4.508 OptimiserWidgetCreateOptDef

Returns an ID referencing the current optimiser widget (GUI)

Arguments

No arguments

Returns

Return type: Real

ID to reference contents of currently highlighted optimiser widget (GUI). Returns -1 if no widget is currently open.

4.509 OptimiserWidgetGetDataObject

Returns data for an optimiser widget (GUI) data object. Data objects are shown in the Data view in the optimiser GUI. Each object has a unique name which must be passed to argument 1.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Name of data object

Argument 1

Name of data object. This can be obtained from [OptimiserWidgetGetDataObjectNames\(\)](#)

Returns

Return type: Real array (XY values)

XY Vector providing values of data object

4.510 OptimiserWidgetGetDataObjectNames

Returns names of all data objects in the current optimiser widget (GUI). Name can be used as an argument to [OptimiserWidgetGetDataObject\(\)](#) to get actual data.

Arguments

No arguments

Returns

Return type: String array

List of names

See Also

[OptimiserGetDataObject\(\)](#)

4.511 OptimiserWidgetGetInfo

Get optimiser widget (GUI) file information

Arguments

No arguments

Returns

Return type: String array

- 0 Path to attached schenmatic
- 1 Path to results file
- 2 Path to definition file

Returns an empty vector if ID invalid

See Also

[OptimiserSimulatorGetInfo\(\)](#)

[OptimiserGetInfo\(\)](#)

4.512 OptimiserWidgetPreFlightChecks

Check optimiser widget (GUI) data prior to starting an optimiser run. Checks that the configuration is valid. The following checks are made:

- Checks parameter initial values are inside any minimum or maximum values
- Checks that minimum and maximum values are defined for algorithms that require them
- Checks that the specified algorithms supports constraints if any are specified
- Checks that SIMPLIS runs do not specify more than one of each analysis type

Checks that a SIMPLIS POP analysis is defined if an AC analysis is also defined

Checks that SIMPLIS runs do not define measurement data for POP analyses if tranalso specified

Checks that there is one and one only objective function

Arguments

No arguments

Returns

Return type: String error

Two element string array.

- 0 1: ok to run 0: not Ok to run
- 1 Error message

4.513 OptimiserWidgetWriteCommandFile

Create SIMetrix simulator command file from current widget (GUI). The optimiser definition describes the parameters, analyses and measurements to run an optimisation analysis. The SIMetrix simulator uses a number of netlist commands to describe the analysis. This function generates those commands from the current widget.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Filename to receive simulator commands

Argument 1

Filename to receive simulator commands

Returns

Return type:

4.514 OptimiserWidgetWriteXML

Writes current optimiser widget to the file specified in the argument

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Filename

Argument 1

Path to file to receive XML definition

Returns

Return type: Real

Return status: Success:1, write failed:0, No widget open:-1

4.515 OptimiserWidgetWriteXMLString

Returns string containing XML definition of current optimiser widget (GUI)

Arguments

No arguments

Returns

Return type: String

String XML definition of optimiser widget. Empty if no optimiser widget currently open

4.516 OptimiserWriteCommandFile

Create SIMetrix simulator command file from optimiser definition. The optimiser definition describes the parameters, analyses and measurements to run an optimisation analysis. The SIMetrix simulator uses a number of netlist commands to describe the analysis. This function generates those commands from an optimiser definition.

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID
2	String	Yes		Filename to receive simulator commands

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)
[OptimiserCreateFromXML\(\)](#)
[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Argument 2

Filename to receive simulator commands

Returns

Return type: Real

Error code:

- 1 ID invalid
- 0 Function completed successfully
- 1 Cannot open file
- 2 Optimiser definition ID invalid
- 3 Cannot open file and Optimiser definition ID invalid
- 4 No analyses defined in definition

4.517 OptimiserWriteHTMLReport

Creates a human-readable HTML report file from an optimiser definition. Usually, the optimiser definition would have been created from the optimiser results file. The results file is an XML file that contains the definition information along with results including the data for each iteration and the final result.

The results file is automatically generated for each optimiser run. The path of the file can be obtained from [OptimiserGetInfo\(\)](#) or [OptimiserSimulatorGetInfo\(\)](#). This file can then be loaded using [OptimiserCreateFromXML\(\)](#) and the resulting ID used for argument 1.

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID
2	String	Yes		Path of file to receive HTML content
3	String	No	An internal default style will be used	CSS file

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)
[OptimiserCreateFromXML\(\)](#)
[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Argument 2

Path of file to receive HTML content

Argument 3

CSS file containing style information for the HTML content.

CSS style names:

datatable	General styles used for table containing iteration data
headertable	General styles used for table at top of report
opt_expression	Style for table column holding the expression
opt_none	Style for text showing values that are defined as report-only
opt_constraint_pass	Style for text showing constraint values that passed
opt_constraint_fail	Style for text showing constraint values that failed

The CSS file may also define h1, h2 and p tags used in the HTML document

Returns

Return type: Real

Status. 1 for success, 0 failure, -1 definition not recognised

4.518 OptimiserWriteXML

Writes out an optimiser definition to a file in XML format. The optimiser definition including its parameters, analyses and measurements can be described using an XML based definition. This function writes out that definition to a file.

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID
2	String	Yes		Filename

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)
[OptimiserCreateFromXML\(\)](#)
[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Argument 2

Filename to receive XML definition

Returns

Return type: Real

Status: 1.0 success, 0.0 save failed, -1.0 ID not recognised

See Also

[OptimiserWriteXMLString\(\)](#)

4.519 OptimiserWriteXMLString

Returns an optimiser XML definition as a string. The optimiser definition including its parameters, analyses and measurements can be described using an XML based definition.

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Optimiser definition ID

Argument 1

ID reference to optimiser definition as returned by functions:

[OptimiserCreateFromXMLString\(\)](#)
[OptimiserCreateFromXML\(\)](#)
[OptimiserWidgetCreateOptDef\(\)](#)
[OptimiserSimulatorGetDef\(\)](#)

Returns

Return type: String

Complete optimiser XML definition. Returns empty if ID invalid

See Also

[OptimiserWriteXML\(\)](#)

[OptimiserSimulatorWriteXMLString\(\)](#)

4.520 Parse

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Input string
2	string	No	Space, tab, comma	Delimiters
3	string	No	<<empty>>	Options

Returns

Return type: string array

4.521 ParseAnalysis

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Analysis spec

Argument 1

Analysis spec as it would appear in a netlist or the F11 window. E.g. lines beginning with .TRAN, .AC, .DC etc.

Returns

Return type: string array

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'
```

4.522 ParseEscape

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. Separators preceded by a backslash (\) in the input string will be escaped and the string will not split on those separators. All escaped separators are replaced by the unescaped separators in the return string array.

For example, the following call to ParseEscape():

```
ParseEscape('A,List,of \, Delimited Items', ',')
```

returns:

```
A
List
of \, Delimited Items
```

Note the return value has the escaped comma separator “\,” replaced with “,”

The default separators are space, comma and tab. User-defined, single character separators can be supplied as the second argument. Separators are case sensitive.

The same string parsed with the default separators of space, comma and tab:

```
ParseEscape('A,List,of \, Delimited Items')
```

returns:

```
A
List
of
,
Delimited
Items
```

This function doesn't return empty strings when two delimiters are found in adjacent positions in the input string. This behavior is the same as the [Parse\(\)](#) function. To preserve empty strings, use the [Scan\(\)](#) or [ScanEscape\(\)](#) functions. Using ParseEscape on a string with two adjacent delimiters:

```
ParseEscape('Two, separators', ', ')
```

returns:

```
Two
separators
```

While the same input string provided to `ScanEscape()`:

```
ScanEscape('Two, separators', ' , ')
```

returns:

```
Two
separators
```

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Input string
2	String	No	Space, tab, comma	Delimiters

Argument 1

Input string

Argument 2

Delimiters

Returns

Return type: string array

The input string split into a string array with the escaped delimiters replaced with unescaped delimiters.

See Also

- [Parse\(\)](#)
- [Scan\(\)](#)
- [ScanEscape\(\)](#)

4.523 ParseLaplace

Parses a Laplace expression to return array of denominator and numerator coefficients

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Laplace expression

Argument 1

Expression in s-variable defining a Laplace transfer function. Refer to the User's Manual -)Parts -)Generic Parts -)Laplace Transfer Function for a detailed explanation .

Returns

Return type: real array

real array as follows:

Index	Description
0	Status code - 0 means success. Use GetLaplaceErrorMessage() to convert this to an error message
1	denominator order
2	numerator order
3 to (3+den order)	denominator coefficients - lowest order first
3+den order+1 to 3+den order+1+num order	numerator coefficients - lowest order first

4.524 ParseParameterString

Legacy function. Use [ParseParameterString2\(\)](#) for new code. 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.

ParseParameterString detects parameter names by searching for known names (as supplied in argument 2) in the supplied string in argument 1. This means that a parameter value that matches a known parameter name may be incorrectly identified as a name. The [ParseParameterString2\(\)](#) function identifies parameter names by their position in the input string and does not suffer from this problem. However [ParseParameterString2\(\)](#) is not compatible in all cases and so ParseParameterString is retained to keep old code working correctly.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		String to parse
2	string	Yes		Parameter names to process
3	string	Yes		action
4	string	No		Write value
5	string	No		Options

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).

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.

Returns

Return type: string array or scalar

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.

4.525 ParseParameterString2

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		String to parse
2	string	Yes		Parameter names to process
3	string	Yes		action
4	string	No		Write value
5	string	No		Options

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.

Returns

Return type: string array or scalar

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.

Example

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')
```

4.526 ParseProbeExpression

Parses an expression used arbitrary fixed probes. The expression uses the access functions V() and I() to denote node voltages and source currents respectively. The function provides a list of the access nodes and sources used in the expression. These are used to create an arbitrary probe symbol.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Expression

Argument 1

Expression to be parsed using V(node) and I(source) access functions

Returns

Return type: string array

Index	Description
0	Expression converted to a form useable by a fixed probe
1	Number of distinct nodes = nn
2	Number of distinct sources = ns
3 to 3+nn-1	Node names used
3+nn to 3+nn+ns-1	Source names used

Index	Description
0	

See Also

[SubstProbeExpression\(\)](#)

4.527 ParseSIMPLISInit

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Simplis init file

Returns

Return type: string array

4.528 ParseSpiceLines

Parses a string in SPICE format and returns each line in a string array. Handles '+' continuation characters by combining to a single line. Whole comment lines (starting with '*') are returned normally but in-line comments using ';' are removed.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Input string

Argument 1

Input string

Returns

Return type: String array

Each element of the array represents a complete SPICE line.

4.529 PathEqual

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. The comparison will be case insensitive.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Path 1
2	string array	Yes		Path 2

Argument 1

First pathname or pathnames to be compared.

Argument 2

Second pathname or pathnames to be compared.

Returns

Return type: real array

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).

4.530 PerCycleTiming

Processes the input vector measuring the Frequency, Period, Duty Cycle, On-Time, or Off-Time on a per-period basis. The returned vector contains the measured value of the input vector, such as the Duty Cycle, plotted against the original x-axis value, for example, time. The return vector is either “stepped” or smooth. A stepped return vector will have vertical discontinuities at the beginning and end of each period found in the input vector, with the value being constant during the input vector period. A smooth return vector will have a single data point per input vector period, located at the mid-point of the input vector period.

Argument 3 is optional and specifies the output curve type, if this argument is not passed, the default value will be “stepped”. The stepped return vector will change value only at the edges detected in the input vector. The value will be constant during the entire period. A smooth input vector will have a single data point at the mid-point of the input vector period. The points will be connected resulting in a smooth curve from one period to the next.

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. This argument is only used for the period and frequency measurements. All other measurements will be processed with the Direction argument set to 1, indicating positive edges.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Input vector
2	string	Yes		Measurement to make
3	string	No	stepped	Type of return curve
4	real	No	1	Direction

Argument 1

The vector to return the Frequency, Period, Duty Cycle, On-Time, or Off-Time values for. The timing edges are found from this vector using a threshold of $\frac{maximum+minimum}{2}$. For this reason, it is important that the vector have a uniform amplitude and is noise-free around the trigger threshold.

Argument 2

A pre-defined measurement function to make, one of:

- frequency
- period
- duty-cycle
- on-time
- off-time

Argument 3

Determines the type of return curve, one of:

- stepped
- smooth

Argument 4

Determines the edges used to process the input vector, one of:

- -1: Falling edges
- 1 : Rising edges
- 0 : Both Rising and Falling Edges

Returns

Return type: real array

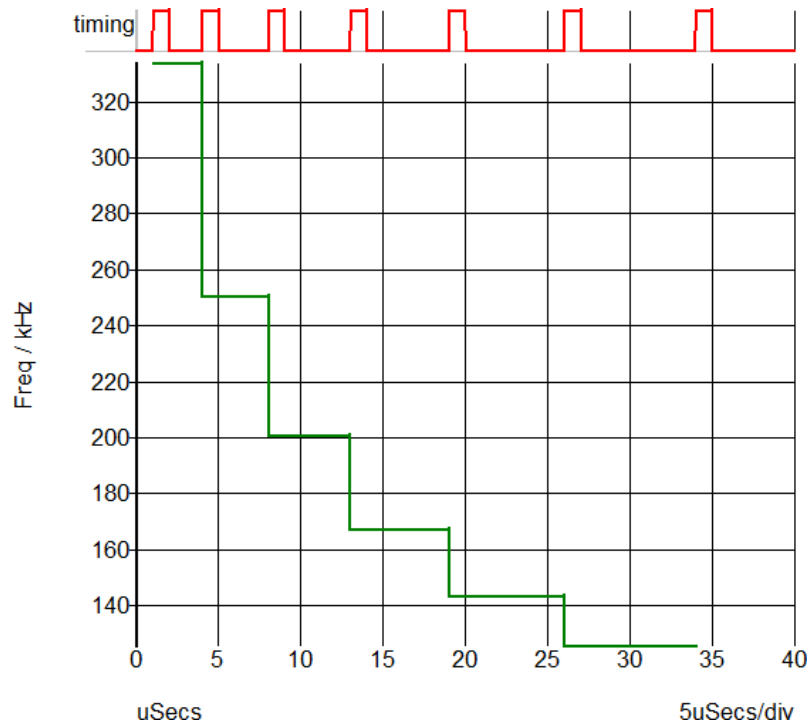
A real vector of the measured values from the input vector, with reference values according to the third argument. The return vector is formatted to be plotted directly on the waveform viewer.

Example

A call to:

```
PerCycleTiming( :Gate , 'frequency' )
```

will generate a vector which, when plotted on the waveform viewer appears like:



4.531 PerCycleValue

Processes the input vector measuring Minimum, Maximum, Mean, Peak-to-Peak, or the RMS value of the input vector during time intervals generated by the timing vector. The returned vector contains the measured value of the input vector, such as the Mean value, plotted against the timing vector x-axis value, for example, time. The return vector is either "stepped" or smooth. A stepped return vector will have vertical discontinuities at the beginning and end of timing vector period. A smooth return vector will have a single data point per input vector period, located at the mid-point of the input vector period.

Argument 4 specifies the output curve type with the default being "stepped". The stepped return vector will change value only at the edges detected in the input vector. The value will be constant during the entire period. A smooth input vector will have a single data point at the mid-point of the input vector period. The points will be connected resulting in a smooth curve from one period to the next.

Argument 5 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. This argument is only used for the period and frequency measurements. All other measurements will be processed with the Direction argument set to 1, indicating positive edges.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Input vector
2	real array	Yes		Input vector
3	string	Yes		Measurement to make
4	string	No	stepped	Type of return curve
5	real	No	1	Direction

Argument 1

The vector to measure the Minimum, Maximum, Mean, Peak-to-Peak, or the RMS values for. The function finds the timing periods based on the timing vector passed as the second argument.

Argument 2

The vector to determine the period information for the vector input in the first argument. It is expected that the vector input to the first argument will contain noise which precludes using the first argument for any timing measurements. To fail. For this reason, the function finds the edges from this vector using a threshold of $\frac{\text{maximum} + \text{minimum}}{2}$. For this reason, it is important that the vector have a uniform amplitude and is noise-free around the trigger threshold.

It is possible that the input vector is free of noise, in which case the same vector could be input to both the first and second function arguments. An example of this would be the output of a gate which has well-defined transitions and uniform maximum and minimum amplitudes.

Argument 3

A pre-defined measurement function to make, one of:

- minimum
- maximum
- mean
- peak-to-peak
- rms

Argument 4

Determines the type of return curve, one of:

- stepped
- smooth

Argument 5

Determines the edges used to process the timing vector, one of:

- -1: Falling edges

- 1 : Rising edges
- 0 : Both Rising and Falling Edges

Returns

Return type: real array

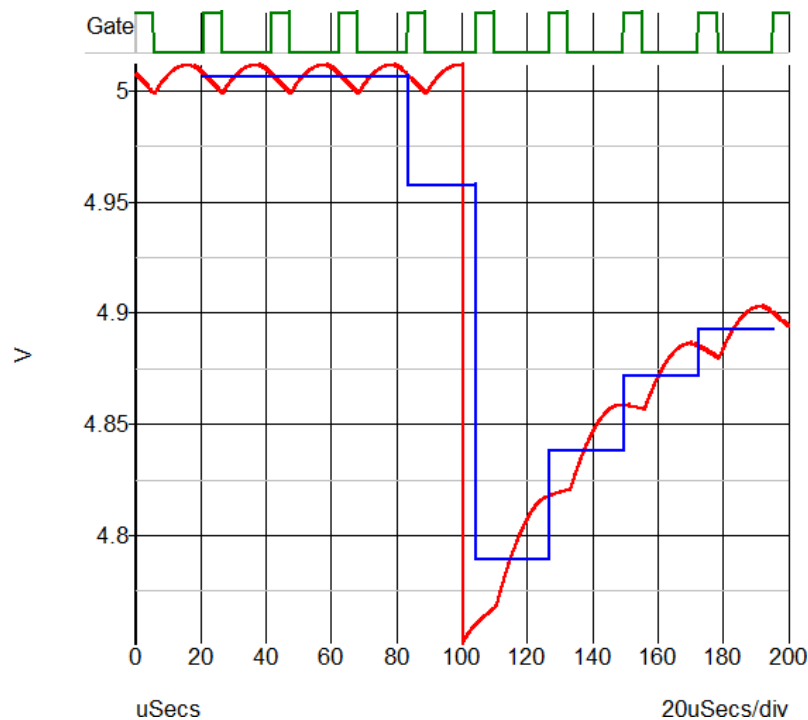
A real vector of the measured values from the input vector, with reference values taken from the timing vector according to the fourth argument. The return vector is formatted to be plotted directly on the waveform viewer.

Example

A call to:

```
PerCycleValue( :Vout , :Clk , 'mean' )
```

will generate a vector which, when plotted on the waveform viewer appears like:



4.532 *ph*

Returns the phase of the argument in degrees.

Each of the functions *ph*, [phase\(\)](#) and [phase_rad](#) produce a continuous output i.e. it does not wrap from 180 degrees to -180 degrees. The [arg\(\)](#) function may be used to obtain a phase value that is always between +/- 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\(\)](#).

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real array

4.533 `phase`

Returns the phase of the argument in degrees. Identical to [ph\(\)](#).

Each of the functions [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](#).

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type:

4.534 `phase_rad`

Returns the phase of the argument in radians. Identical to [ph\(\)](#), except the result is in radians.

Produces a continuous output i.e. it does not wrap from 180 degrees to -180 degrees.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real array

4.535 PhysType

Returns the physical type of the argument.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		Vector

Argument 1

Possible values are.

- '' (meaning dimensionless quantity)
- 'unknown'
- 'Voltage'
- 'Current'
- 'Time'
- 'Frequency'
- 'Resistance'
- 'Conductance'
- 'Capacitance'
- 'Inductance'
- 'Energy'
- 'Power'
- 'Charge'
- 'Flux'
- 'Volt²'
- 'Volt²/Hz'
- 'Volt/rtHz'
- 'Amp²'
- 'Amp²/Hz'
- 'Amp/rtHz'
- 'Volts/sec'

Returns

Return type: string

See Also

[Units\(\)](#)

4.536 PinName

Returns information about the schematic instance pin nearest the mouse cursor. The format of the result depends on the values of the arguments.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	<<empty>>	Options
2	string	No	<<empty>>	Property name

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.

Returns

Return type: string array

4.537 PrepareSetComponentValue

Configures [SetComponentValue\(\)](#) function to define how parameters are stored on schematic instances. The definition is in the form of two tables, 'parameter definitions' and 'implicit defaults'. The 'parameter definitions' defines how parameters are stored. The 'implicit defaults' defines parts that have an implicit value. For example, a resistor value can be set by simply defining the reference of the device without a parameter name. This is known as an implicit value.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Table data
2	string	Yes		Table type

Argument 1

This is either the 'parameter definitions' or the 'implicit defaults' depending on the setting of argument 2. Usually PrepareSetComponentValue is called twice, once for the 'parameter definitions' and again for the 'implicit defaults'.

The 'parameter definitions' table is a List of semi-colon delimited definitions to describe how to handle parameters stored in K=V pairs - as opposed to individual properties. The system looks at the VALUESCRIPT property and its arguments. It scans down the table until it finds an entry that matches the script called by VALUESCRIPT. VALUESCRIPT is the property used by nearly all parts that defines the script that is used to edit the part.

The following table describes the 'parameter definitions' table:

Field number	Description
1	'writeprop' OR 'defaultnames'. If 'writeprop' definition defines the name of the property that will hold the modified K=V values. If 'defaultnames', definition defines how you obtain the names of the parameters and their default values.
2	VALUESCRIPT script name
3	VALUESCRIPT argument to examine. 0 means the VALUESCRIPT arguments are ignored
4	'Direct' OR 'Model'. Only relevant if Field 1 is 'defaultnames'. 'Direct' means the default names data is read from the property specified as the argument in Field 3 or its default in Field 6 (see below). 'Model' means it is read from the params: or vars: list in the device's model file.
5	Boolean can be true/false, off/on or yes/no. Specifies whether the F11 window can be searched for the model. Only relevant if Field 4 is set to 'Model'
6	Default value for argument. If the argument to VALUESCRIPT is not present (or if Field 3 is zero) use this value instead
7	Boolean. Means that a property of the same name will also be written as well as the K=V parameter

The following table describes the 'implicit defaults' table:

Field number	Description
1	Property, Value pair. The value can use wildcards * and ?
2	The property or parameter that is read or set by an implicit action on this device. What happens is that the address is appended with this value when a match is found. So if the user entered U1.R1 where R1 is a resistor, the action will be the same as entering U1.R1.<contents-of-field2>. (And that is how this is implemented internally)

Field number	Description
3	Boolean: If true read or write the first unlabelled value only and leave the rest alone

Argument 2

Specifies what the contents of argument 1 defines. Either 'parameter_definitions' or 'implicit_defaults'

Returns

Return type: real

Number of table entries entered

4.538 PreProcessNetlist

Preprocesses netlist and returns filename where preprocessed result is placed.

Function performs the same task as the PreProcessNetlist command. See [PreProcessNetlist](#)

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Input file name
2	string array			Options

Argument 1

Input file name to be preprocessed

Argument 2

Options

inAppend	Add extra lines separated by semi-colons
simulator	SIMetrix or SIMPLIS. Default is SIMPLIS
mc	Performing a Monte Carlo run
importGlobals	Import global parameter values
params	Provide list of parameters in name=value pairs
mcseed	Monte Carlo seed value
rawdeck	Create raw deck
mclogfile	Create Monte Carlo log file
cpu	Index specifying cpu for multi-core runs
	Used only for decorating the output file name

decoratedeckname Decorate file name according to cpu, parameters and mc seed
 If resulting name is too long, a unique hash value will be used

Returns

Return type: string

File name where preprocessed result is saved

4.539 Probe

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 [Cross probing](#) for an example of using the Probe function.

Arguments

No arguments

Returns

Return type: real

1 if left button clicked, 0 if cancelled (right button or escape)

4.540 ProcessingAccelerator

Detects if the current script was called by an accelerator key

Arguments

No arguments

Returns

Return type: real

1 if the current script was called by an accelerator key, otherwise 0

See Also

[ProcessingDragAndDrop\(\)](#)

[ProcessingGuiAction\(\)](#)

[CommandStatus\(\)](#)

4.541 ProcessingDragAndDrop

Detects if the current script was called by a drag and drop operation

Arguments

No arguments

Returns

Return type: real

1 if current script was called as a result of a drag and drop operation, otherwise 0

See Also

[ProcessingAccelerator\(\)](#)

[ProcessingGuiAction\(\)](#)

[CommandStatus\(\)](#)

4.542 ProcessingGuiAction

Detects if the current script was called by a GUI action. Most scripts are called from a GUI action such as a menu or key press. Typing in the name of the scripts at the command line is also classed as a GUI action. This function will return 1 for such calls.

Scripts can also be called remotely using the /s switch on the SIMetrix.exe command line and also using the SxCommand utility. Such calls are classed as non-GUI. This function will return 0 for such calls.

Arguments

No arguments

Returns

Return type: real

1 if the current script was called by a GUI action, otherwise 0

See Also

[ProcessingAccelerator\(\)](#)

[ProcessingDragAndDrop\(\)](#)

[CommandStatus\(\)](#)

4.543 Progress

Opens a dialog box showing a progress bar.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Position of progress bar in %
2	string array	No	<<empty>>	options/control

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

Returns

Return type: real

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.

4.544 PropFlags

Returns the attribute flags for instances identified by arguments 2 and 3. See "Attribute Flags" in the command [Prop](#) for details. This function has been superseded by [PropFlags2\(\)](#) 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 be provided. It also has more convenient behaviour in the situation when there is no instance match.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name (for flags)
2	string	No	Selected components	Property name (for id)
3	string	No	Instances with property name in arg 2	Property value (for id)
4	real	No	-1	Schematic ID

Argument 1

Property whose flags are to be returned.

Argument 2

Along with argument 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 3

See argument 2.

Argument 4

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string array

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 if the specified schematic could not be found. If no instance matches arguments 2 and 3, an empty *string* will be returned.

4.545 PropFlags2

Returns the attribute flags for instances identified by arguments 3 and 4. See “Attribute flags” in the command [Prop](#) 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name (for flags)
2	real	No	-1	Schematic ID
3	string	No	Selected components	Property name (for id)
4	string	No	Instances with property name in arg 2	Property value (for id)

Argument 1

Property whose flags are to be returned.

Argument 2

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Argument 3

Along with argument 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.

Argument 4

See argument 3.

Returns

Return type: string array

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 the functions [PropValues2\(\)](#) and [SymbolNames\(\)](#) 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* 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.

4.546 PropFlagsAll

Returns the flags for the requested property. This will search all selected elements within a schematic. There are optional filters for choosing elements with a particular property, or property and value combination, along with options to select a specific schematic.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name to retrieve flags for
2	string	No		Filter property name
3	string	No		Filter property value
4	real	No	-1	Schematic ID

Argument 1

The name of the property to return the flags for.

Argument 2

If set, will only select elements that have this property in them.

Argument 3

If set, will only select elements that have the property stated by argument 2, with the value stated by this argument.

Argument 4

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string array

Returns the property flags for all applicable properties. Each row of the resulting array will be a different element's property flag.

Example

The following would return all of the flags for the `ref` property with the selected schematic, for elements that have the property `MODEL` set to `X`:

```
PropFlagsAll(`ref`,`model`,`X`)
```

See Also

[PropFlagsAnnotations\(\)](#)

[PropFlagsWires\(\)](#)

4.547 PropFlagsAnnotations

Returns the flags for the requested property. This will search selected annotations only within a schematic. There are optional filters for choosing elements with a particular property, or property and value combination, along with options to select a specific schematic.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name to retrieve flags for
2	string	No		Filter property name
3	string	No		Filter property value
4	real	No	-1	Schematic ID

Argument 1

The name of the property that we are returning the flags for.

Argument 2

If set, will only select elements that have this property in them.

Argument 3

If set, will only select elements that have the property stated by argument 2, with the value stated by this argument.

Argument 4

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string array

Returns the property flags for all applicable properties. Each row of the resulting array will be a different element's property flag.

See Also

[PropFlagsAll\(\)](#)

[PropFlagsWires\(\)](#)

4.548 PropFlagsWires

Returns the flags for the requested property. This will search selected wires only within a schematic. There are optional filters for choosing elements with a particular property, or property and value combination, along with options to select a specific schematic.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name to retrieve flags for
2	string	No		Filter property name
3	string	No		Filter property value
4	real	No	-1	Schematic ID

Argument 1

The name of the property that we are returning the flags for.

Argument 2

If set, will only select elements that have this property in them.

Argument 3

If set, will only select elements that have the property stated by argument 2, with the value stated by this argument.

Argument 4

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string array

Returns the property flags for all applicable properties. Each row of the resulting array will be a different element's property flag.

See Also

[PropFlagsAll\(\)](#)

[PropFlagsAnnotations\(\)](#)

4.549 PropOverrideStyle

Returns the override style of the selected property, if one exists. Override styles are used in the schematic and symbol editors to assign a different font style to a property. Uses the currently selected schematic/symbol editor.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name
2	string	No		Options

Argument 2

Set to "pin" to declare property name as a pin name

Returns

Return type: string

The override style name, if any, used by the property with the name specified.

4.550 PropValue

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name

Argument 1

Property name

Returns

Return type: string

4.551 PropValues

Returns a property value for instances identified by arguments 2 and 3.

This function has been superseded by [PropValues2\(\)](#) 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name whose value is required
2	string	No	Use selected components if omitted	Property name to identify instance
3	string	No	All instances with property name in arg2	Property value to identify instance
4	real	No	-1	Schematic ID

Argument 1

Property name whose value is required

Argument 2

Along with argument 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 3

See argument 2.

Argument 4

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string array

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 if the specified schematic could not be found. If no instance matches arguments 2 and 3, an empty *string* will be returned.

4.552 PropValues2

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name whose value is required
2	real array	No	-1	Schematic handle and options
3	string	No	Use selected components if omitted	Property name to identify instance
4	string	No	All instances with property name in arg2	Property value to identify instance

Argument 1

Property whose value is to be returned.

Argument 2

First element is a schematic handle as returned by the function [OpenSchematic\(\)](#). 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.

If a third element is present and set to a non-zero value, symbolic properties will be resolved.

Argument 3

Along with argument 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.

Argument 4

See argument 3.

Returns

Return type: string array

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\(\)](#) and [SymbolNames\(\)](#) 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.

4.553 PropValuesAll

Returns the values for the requested property. This will search all selected elements within a schematic. There are optional filters for choosing elements with a particular property, or property and value combination, along with options to select a specific schematic.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name to retrieve values for
2	string	No		Filter property name
3	string	No		Filter property value
4	real	No		Schematic handle

Argument 1

The name of the property to return the values for.

Argument 2

If set, will only choose elements that have this property in them.

Argument 3

If set, will only choose elements that have the property stated by argument 2, with the value stated by this argument.

Argument 4

Handle to a particular schematic. If not set, uses the currently highlighted schematic.

Returns

Return type: string array

Returns the property values for all applicable properties. Each row of the resulting array will be a different element's property flag.

Example

The following would return all of the value for the `ref` property with the selected schematic, for elements that have the property `MODEL` set to `X`:

```
PropValuesAll(`ref`,`model`,`X`)
```

See Also

[PropValuesAnnotations\(\)](#)

[PropValuesWires\(\)](#)

4.554 PropValuesAnnotations

Returns the values for the requested property. This will search selected annotations only within a schematic. There are optional filters for choosing elements with a particular property, or property and value combination, along with options to select a specific schematic.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name to retrieve values for
2	string	No		Filter property name
3	string	No		Filter property value
4	real	No		Schematic handle

Argument 1

The name of the property to return the values for.

Argument 2

If set, will only choose elements that have this property in them.

Argument 3

If set, will only choose elements that have the property stated by argument 2, with the value stated by this argument.

Argument 4

Handle to a particular schematic. If not set, uses the currently highlighted schematic.

Returns

Return type: string array

Returns the property values for all applicable properties. Each row of the resulting array will be a different element's property flag.

See Also

[PropValuesAll\(\)](#)

[PropValuesWires\(\)](#)

4.555 PropValuesWires

Returns the values for the requested property. This will search selected wires only within a schematic. There are optional filters for choosing elements with a particular property, or property and value combination, along with options to select a specific schematic.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name to retrieve values for
2	string	No		Filter property name
3	string	No		Filter property value
4	real	No		Schematic handle

Argument 1

The name of the property to return the values for.

Argument 2

If set, will only choose elements that have this property in them.

Argument 3

If set, will only choose elements that have the property stated by argument 2, with the value stated by this argument.

Argument 4

Handle to a particular schematic. If not set, uses the currently highlighted schematic.

Returns

Return type: string array

Returns the property values for all applicable properties. Each row of the resulting array will be a different element's property flag.

See Also

[PropValuesAll\(\)](#)

[PropValuesAnnotations\(\)](#)

4.556 PutEnvVar

Write a system environment variable. Note that this only modifies environment variables in the current process and any child processes initiated using the commands [Shell](#) or [ShellOld](#).

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Definition

Argument 1

Definition. Must be of form `name=value`.

Returns

Return type: real

The function returns 1 on success or 0 on failure. Failure can occur if the argument is of the wrong format.

4.557 PWLCurveFit

Returns a real vector with a Piecewise Linear Approximation of the input real array. The number of PWL segments is determined by the second argument. See the argument definitions for specialized curve fits and additional options.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		The XY input vector
2	real	Yes		The number of segments
3	string	No		Specifies the use of a specialized algorithm
4	real	No		Fix the last PWL point?
5	real	No		Optional parameters

Argument 1

The vector to process. Must have a reference (x values).

Argument 2

The number of segments of the resulting piecewise linear approximation.

Argument 3

Sets the type of algorithm to use. Can be 'generic', 'diode', 'capacitor', or 'inductor'.

Argument 4

Fixes the last PWL point to be the last data point of the input vector

Argument 5

Optional parameters for the 'generic', 'diode', and 'inductor' algorithms. For 'diode', a number (resistance) greater than 0 will force the first segment to have a slope of 1/resistance. 0 will force the first segment to have a slope of 0. For 'inductor', 1 will symmetrically fit the curve around (0,0). For 'generic', 1 will curve fit through point (0,0).

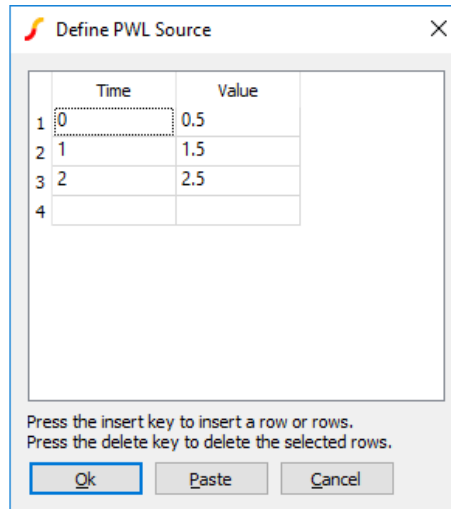
Returns

Return type: real array

—

4.558 PWLDialog

Opens the dialog box shown below allowing the entry of X-Y pairs intended for the definition of piece-wise linear devices.



Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		X-Y Pairs
2	string	No		Options
3	real array	No		Initial condition / value states
4	string	No		Conversion tool entry history

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 seven 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'

Index	Description
3	Initial condition mode. May be: ‘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.
4	Help context id. Default: ‘-1’ (no help button shown)
5	Minimum number of segments. Default = ‘1’
6	Maximum number of segments. Default = ‘255’
7	Symmetric definition flag. ‘1’ enables symmetric definition mode. Default ‘0’.
8	Enable repeat function. ‘1’ enables repeat options used for signal source. The repeat function cannot be displayed at the same time as the initial condition options

Argument 3

Real array with up to four 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. Third element if defined sets the state of the ‘Idle in POP’ check box. Fourth element defines the number of repeat cycles. If set to zero the ‘Repeat forever’ button is checked.

Argument 4

A string containing the unit input and energy storage output history of the last conversion tool entry. The input and output X-Y points are concatenated with spaces and are then concatenated together with a ‘|’ character.

Returns

Return type: string array

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.

If the repeat function is enabled, the number of repeat cycles will be the final element; however, if the component is a capacitor or inductor, the unit-to-energy storage conversion tool entry history will be the final element.

4.559 QueryData

Filters a list of data items according to search criteria.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Data
2	string array	Yes		Filter

Argument 1

The data to be filtered. This should consist of an array of strings comprising semicolon 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'.

Returns

Return type: string array

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.

4.560 RadioSelect

Opens a dialog box with any number of radio buttons. The number of buttons visible depends on the length of argument 2. Six will be displayed if it is omitted.

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	1	Number of buttons initially selected
2	string	No	empty	Button labels
3	string	No	Dialog box caption	Other labels
4	string	No		Help context ID

Argument 1

The number of buttons initially selected.

Argument 2

Specifies the labels for each button.

Argument 3

String array up to length 3. First element is dialog box caption and the second element is text label displayed above radio buttons. If a third element is present, a check box will also be displayed underneath the radio buttons. The third element defines the label for this check box

Argument 4

Specifies a help context id and if present a Help button will be displayed. This is used by some internal scripts.

Returns

Return type: real

The return value identifies the selected button with the top most being 1. If the user cancels the function returns 0. If the check box is displayed, the return value will have length 2 with the second element holding the state of the check box.

See Also

[BoolSelect\(\)](#)

[EditSelect\(\)](#)

[ValueDialog\(\)](#)

[NewValueDialog\(\)](#)

4.561 Range

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 function [Truncate\(\)](#).

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex or string array	Yes		Vector
2	real	No		Start index
3	real	No	Vector length -1	End index

Returns

Return type: matches argument 1

4.562 re

Returns the real part of the complex argument.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real array

Returns the real part of the complex argument.

4.563 ReadClipboard

Returns text contents of the windows clipboard. Data is returned as one line per array element.

Arguments

No arguments

Returns

Return type: string array

4.564 ReadConfigCollection

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 the function [ReadConfigSetting\(\)](#) with an empty second argument.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Section

Argument 1

Name of section to return.

Returns

Return type: string array

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 the function [AddConfigCollection\(\)](#) to write values to the list.

4.565 ReadConfigSetting

Reads a configuration setting. Configuration settings are stored in the configuration file. See *User's Manual/Sundry Topics/Configuration Settings* 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 function [GetOption\(\)](#).

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Section
2	string	No		Key

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.

Returns

Return type: string or string array

Value read from configuration file.

See Also

[WriteConfigSetting\(\)](#)

4.566 ReadF11Analyses

Reads SIMetrix simulator analysis specifications in the schematic F11 window and returns a string array describing parameters for a specified analysis type

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Analysis type

Argument 1

Define the type of analysis to be returned. Maybe one of the following string values

AC	AC analysis
DC	DC analysis
NOISE	Noise analysis
TF	Transfer function analysis
TRAN	Transient analysis
ACMULTISTEP	AC multistep analysis
DCMULTISTEP	DC multistep analysis
NOISEMULTISTEP	Noise multistep analysis
TFMULTISTEP	Transfer function multistep analysis
TRANMULTISTEP	Transient multistep analysis

Returns

Return type: string array

Three element string array.

- 0 Indicates whether analysis type is defined
‘0’: not defined, ‘1’: defined
- 1 Indicates whether analysis type is enabled
‘0’: not enabled, ‘1’: enabled
Disabled analysis definitions are prefixed with a single ‘*’
- 2 List of name=value pairs describing various analysis parameters

4.567 ReadF11Options

Read .OPTIONS line in the F11 window

Arguments

No arguments

Returns

Return type: string array

Array of semi-colon delimited strings providing details of any SIMetrix .OPTIONS statements located in the current schematic’s F11 window. Each token in the string is defined in the following table:

Field	Description
0	Option name
1	Value
2	Type - on eof 'BOOL', 'REAL', 'INT', 'STRING' or 'UNKNOWN'

The function will not return option settings that are not recognised by the simulator. It will also not return option settings that are set to their default value.

See Also

[WriteF11Options\(\)](#)

[WriteF11Lines\(\)](#)

[GetF11Lines\(\)](#)

[AppendTextWindow](#)

4.568 ReadFile

Returns an array of strings holding lines of text from the file specified by argument 1.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File name
2	string	Yes	'utf8'	Encoding option

Argument 2

Can be 'mbcs' or 'utf8'. If 'utf8' the file is assumed to be encoded using UTF8. If 'mbcs' encoding uses the system default

Returns

Return type: string array

See Also

[LoadFile\(\)](#) Performs a similar operation but with more encoding options including the ability to auto-detect UTF8

4.569 ReadIniKey

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Inifile name
2	string	Yes		Section name
3	string	Yes		Key name

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.

Returns

Return type: string array

string array

4.570 ReadRegSetting

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Key name
2	string	Yes		Value name
3	string	No	'HKCU'	Top level tree

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'.

Returns

Return type: string

Returns value read from the registry. If the value doesn't exist, the function returns an empty vector.

4.571 ReadSchemProp

Returns value of schematic window property value.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name
2	string	No	Currently displayed	Schematic path
3	number	No		Schematic handle

Argument 1

Property name. There are a number of built-in properties that are always available. Others can be created with the function [WriteSchemProp\(\)](#). The 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 6th box from the left.
'UserVersion'	Read-only. Current version number of schematic. This is updated each time the schematic is saved
'ID'	Read-only. Returns ID of schematic (same value returned by OpenSchematic())
'Magnification'	Read-only. Current view magnification
'Modified'	Modified status 'TRUE' or 'FALSE'

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.

Argument 3

Schematic handle.

Returns

Return type: string

Returns the value of the property

4.572 ReadSIMPLISF11Data

Returns information about the current SIMPLIS simulation configuration stored in the schematic's F11 window.

Arguments

No arguments

Returns

Return type: string array

Array of semi-colon delimited strings. Each string defines a SIMPLIS simulation option setting. The first field is the name of the setting. The second is the type (boolean, integer, real, enum) and the third is the current value.

4.573 ReadSpiceFile

Returns an array of strings holding lines of text from the file specified by argument 1. Continuation lines marked with a '+' are merged to a single line and inline comments are removed. Comment lines starting with a '*' are included in the output.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File name

Returns

Return type: string array

4.574 ReadTextEditorProp

Reads a text editor property. This will work for all text based editors.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name
2	string	No		Editor type

Argument 1

Name of the property to read the value for.

Argument 2

Optional flag to specify the type of editor. Possible values are:

- LogicDefinitionEditor
- NetlistEditor
- ScriptEditor
- TextEditor
- VerilogAEditor
- VerilogHDLEditor

Returns

Return type: string

The property value for the requested property.

4.575 ReadTouchstone

Retrieves the data from a previously Touchstone file loaded using [LoadTouchstone\(\)](#). Must provide data ID, row index and column index

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Data ID returned by LoadTouchstone()
2	Real	Yes		Row index
3	Real	Yes		Column index

Argument 1

Data ID returned by [LoadTouchstone\(\)](#)

Argument 2

Row index

Argument 3

Column index

Returns

Return type: Real array

Y-parameter data in triplets. First value is frequency, second value is real part, third value is imaginary part. One triplet is returned for each frequency found in the file

4.576 *real*

Returns the real part of the complex argument.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real array

Returns the real part of the complex argument.

4.577 *Ref*

Returns the reference of the argument. See [Vector References](#).

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		Vector

Returns

Return type: real/complex array

4.578 RefName

Returns the name of the reference of the supplied vector. See [Vector References](#). Note that the function [Ref\(\)](#) returns the actual data for the reference.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		Vector

Returns

Return type: string

4.579 RegExContains

This function compiles a regular expression from the second argument and searches for a regular expression match in the string supplied as the first argument. The return value is a 1 if one or more matches of the regular expression exist, 0 if no matches are found in the input string.

The third argument is the offset into the input string to start the regular expression search.

In common with the other Regular Expression functions, this function is case *insensitive* by default. Also by default, the dot metacharacter character '.' does not match new line characters. You can change these behaviors using two options passed as the fourth argument.

The regular expression functions in SIMetrix/SIMPLIS support Perl-compatible regular expressions. Forward references, back references, alternation, capturing up to 99 groups, negative and positive look behind and ahead assertions are all supported. Regular expressions can be tested for validity with the [RegExIsValid\(\)](#) function.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		String to search
2	string	Yes		Regular expression
3	real	No	0	Character offset
4	string	No		Options

Argument 1

The input string to test if there is one or more matches to the regular expression.

Argument 2

The regular expression string.

Argument 3

The character position to start the regular expression search. 0 starts the search at the beginning of the input string.

Argument 4

Two options are supported:

- `casesensitive`
- `dotmatchesall`

Passing the first option `'casesensitive'` makes the regular expression match case sensitive. The second option `'dotmatchesall'` causes the dot (`'.'`) metacharacter to match new lines in the input string. You can pass both options as a string array:

```
[ 'casesensitive' , 'dotmatchesall' ]
```

Returns

Return type: real

A real number 1, indicating one or more matches of the regular expression are found in the input string, or 0 indicating no matches of the regular expression are found in the input string.

Example

Find if a string starts with a certain phrase:

```
RegexContains( 'Abc' , '^ab' )
```

returns 1 because `'Abc'` starts with `'ab'` without case sensitivity.

Setting case sensitive matching, starting at the beginning of the input string:

```
RegexContains( 'Abc' , '^ab' , 0 , 'casesensitive' )
```

returns 0 because the case sensitive match fails.

See Also

- [RegexIsValid\(\)](#)
- [RegexMatch\(\)](#)
- [RegexReplace\(\)](#)
- [RegexSearch\(\)](#)
- [RegexSearchAll\(\)](#)
- [RegexStrStr\(\)](#)
- [RegexSplit\(\)](#)

4.580 RegExIsValid

This function checks if the regular expression provided as the first argument is a valid regular expression. There are several reasons why a regular expression could be invalid. These include unbalanced capturing parentheses, referencing a captured group which is not actually captured etc. This function returns a real 1 if the regular expression is valid or a string with an error message. You can test the type of the return value with the [IsStr\(\)](#) function.

Note that you will only need to use this function for development or in a script where the regular expression is created during script execution. Any regular expression entered as a string literal can be checked before the script is executed using this function, and the actual script will not need to call this function.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Regular expression

Argument 1

The regular expression string.

Returns

Return type: real or string

If the expression is valid, the return is a real 1. If invalid, a string error message describing why the regular expression is not valid.

Example

An example of a missing opening parentheses:

```
RegExIsValid('asdfghjkl')
```

returns 'Invalid regular expression (unmatched closing parenthesis) index into regular expression string where error occurs : 9'

An example of a missing closing parentheses:

```
RegExIsValid('( asdfghjkl')
```

returns 'Invalid regular expression (missing closing parenthesis) index into regular expression string where error occurs : 10'

Once the opening and closing parentheses are correctly matched:

```
RegExIsValid('( asdfghjkl)')
```

returns 1.

See Also

- [RegExContains\(\)](#)
- [RegExMatch\(\)](#)
- [RegExReplace\(\)](#)
- [RegExSearch\(\)](#)
- [RegExSearchAll\(\)](#)
- [RegExStrStr\(\)](#)
- [RegExSplit\(\)](#)

4.581 RegExMatch

This function compiles a regular expression from the second argument and searches for a regular expression match in the string supplied as the first argument. The return value is a string array containing the matched strings. This first index is the string which the entire regular expression matched, each index after the first represents any captured text using the capturing groups syntax: (). If no match occurs, an empty vector is returned.

The third argument is the offset into the input string to start the regular expression search.

In common with the other Regular Expression functions, this function is case *insensitive* by default. Also by default, the dot metacharacter character '.' does not match new line characters. You can change these behaviors using two options passed as the fourth argument.

The regular expression functions in SIMetrix/SIMPLIS support Perl-compatible regular expressions. Forward references, back references, alternation, capturing up to 99 groups, negative and positive look behind and ahead assertions are all supported. Regular expressions can be tested for validity with the [RegExIsValid\(\)](#) function.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		String to search
2	string	Yes		Regular expression
3	real	No	0	Character offset
4	string	No		Options

Argument 1

The input string to find the index where the regular expression match occurs.

Argument 2

The regular expression string.

Argument 3

The character position to start the regular expression search. 0 starts the search at the beginning of the input string.

Argument 4

Two options are supported :

- `casesensitive`
- `dotmatchesall`

Passing the first option 'casesensitive' makes the regular expression match case sensitive. The second option 'dotmatchesall' causes the dot('.') metacharacter to match new lines in the input string. You can pass both options as a string array:

```
[ `casesensitive' , `dotmatchesall' ]
```

Returns

Return type: string array

String array containing the matched strings.

Example

A simple example without capturing groups:

```
RegExMatch(`abcdef`, `de|bc` )
```

returns the matched string, in this case 'bc'.

A more advanced example where substrings are found and captured:

```
RegExMatch( 'POWER_SUPPLY_LOAD_3_RES' , '(SIMPLIS_DVM_ADVANCED|POWER_SUPPLY)_LOAD_(\d)' )
```

returns a string array with three elements. The first element contains the entire matched text, the second element contains the string matched by the first captured group (SIMPLIS_DVM_ADVANCED|POWER_SUPPLY), the third element contains the text from the second captured group (3):

```
'POWER_SUPPLY_LOAD_3'  
'POWER_SUPPLY'  
'3'
```

A failing match:

```
RegExMatch( 'res_z' , '(SIMPLIS_DVM_ADVANCED|POWER_SUPPLY)_LOAD_(\d)' )
```

returns an empty vector. You can check the length of the return with the [Length\(\)](#) function.

See Also

- [RegExIsValid\(\)](#)
- [RegExContains\(\)](#)
- [RegExReplace\(\)](#)

- [RegExpSearch\(\)](#)
- [RegExpSearchAll\(\)](#)
- [RegExpStrStr\(\)](#)
- [RegExpSplit\(\)](#)

4.582 **RegExp**

Match an input string to a sequence of regular expressions. The first expression is matched to to the input string and the result returned in the first element of the return array. If the expression does not match from the first character of the input string, the pattern matching will advance to subsequent characters until a match is found. The process is then repeated for subsequent expressions until all have been processed or no match is found.

Arguments

Number	Type	Compulsory	Default	Description
1	String array	Yes		Array of regular expressions
2	String	Yes		Input string

Argument 1

Array of regular expressions. The input string is matched in sequence

Argument 2

Input string

Returns

Return type: String array

First element is either 'CompleteMatch' or 'IncompleteMatch'. 'CompleteMatch' will be returned if all regular expressions are matched to the input string without skipping any characters.

Subsequent fields contain the parts of the string that match the corresponding regular expression.

4.583 **RegExpReplace**

This function compiles a regular expression from the second argument and searches for a regular expression match in the string supplied as the first argument. Where the regular expression matches, the replacement string is inserted into the input string. The return value is the input string with the replaced matches. If no match occurs, the input string is returned unaltered.

The third argument is the replacement string, which can contain backreferences.

The fourth argument is the offset into the input string to start the regular expression search.

In common with the other Regular Expression functions, this function is case *insensitive* by default. Also by default, the dot metacharacter character '.' does not match new line characters. You can change these behaviors using two options passed as the fifth argument.

The regular expression functions in SIMetrix/SIMPLIS support Perl-compatible regular expressions. Forward references, back references, alternation, capturing up to 99 groups, negative and positive look behind and ahead assertions are all supported. Regular expressions can be tested for validity with the [RegExIsValid\(\)](#) function.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		String to search
2	string	Yes		Regular expression
3	string	Yes		Replacement string
4	real	No	0	Character offset
5	string	No		Options

Argument 1

The input string to find and replace where the regular expression match occurs.

Argument 2

The regular expression string.

Argument 3

The replacement string, which can contain backreferences.

Argument 4

The character position to start the regular expression search. 0 starts the search at the beginning of the input string.

Argument 5

Two options are supported :

- `casesensitive`
- `dotmatchesall`

Passing the first option '`casesensitive`' makes the regular expression match case sensitive. The second option '`dotmatchesall`' causes the dot('.') metacharacter to match new lines in the input string. You can pass both options as a string array:

```
[ `casesensitive' , `dotmatchesall' ]
```

Returns

Return type: string

The input string with the regular expression replacements

Example

A simple example where the number of pins is replaced with '3':

```
RegexReplace( 'SIMPLIS_DVM_ADVANCED_LOAD_4T_RES_BODE', '\d' , '3')
```

returns 'SIMPLIS_DVM_ADVANCED_LOAD_3T_RES_BODE'.

A more advanced example which produces the same result but a much more restrictive regular expression, and the second backreference.

```
RegexReplace( 'SIMPLIS_DVM_ADVANCED_LOAD_4T_RES_BODE', '^SIMPLIS_DVM_ADVANCED_LOAD_(\d)T(.*?)$', '\1\2')
```

returns 'SIMPLIS_DVM_ADVANCED_LOAD_3T_RES_BODE'.

A failing match and replace:

```
RegexReplace( 'res_z' , '^SIMPLIS_DVM_ADVANCED_LOAD_(\d)T(.*?)$', 'SIMPLIS_DVM_ADVANCED_LOAD_3T_RES_BODE')
```

Returns the input string: 'res_z'.

See Also

- [RegexIsValid\(\)](#)
- [RegexContains\(\)](#)
- [RegexMatch\(\)](#)
- [RegexSearch\(\)](#)
- [RegexSearchAll\(\)](#)
- [RegexStrStr\(\)](#)
- [RegexSplit\(\)](#)

4.584 **RegexSearch**

This function compiles a regular expression from the second argument and searches for a regular expression match in each element of the string array supplied as the first argument. The return value is the index into the input string array where the first match occurs, or -1 if no matches are found. This function is a regular expression version of the [Search\(\)](#) function.

The third argument is the offset into the input string to start the regular expression search.

In common with the other Regular Expression functions, this function is case *insensitive* by default. Also by default, the dot metacharacter character '.' does not match new line characters. You can change these behaviors using two options passed as the fourth argument.

The regular expression functions in SIMetrix/SIMPLIS support Perl-compatible regular expressions. Forward references, back references, alternation, capturing up to 99 groups, negative and positive look behind and ahead assertions are all supported. Regular expressions can be tested for validity with the [RegexIsValid\(\)](#) function.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		String array to search
2	string	Yes		Regular expression
3	real	No	0	Character offset
4	string	No		Options

Argument 1

The input string array to search for the regular expression match.

Argument 2

The regular expression string.

Argument 3

The character position to start the regular expression search. 0 starts the search at the beginning of the input string.

Argument 4

Two options are supported:

- casesensitive
- dotmatchesall

Passing the first option 'casesensitive' makes the regular expression match case sensitive. The second option 'dotmatchesall' causes the dot ('.') metacharacter to match new lines in the input string. You can pass both options as a string array:

```
[ `casesensitive' , `dotmatchesall' ]
```

Returns

Return type: real

The index into the input string array where the first match occurs, or -1 indicating no matches of the regular expression are found in the input string array.

Example

A simple example where a string array is searched for a certain phrase:

```
RegExSearch( [ '.tran lm 0' , '.pop TRIG_COND=1_TO_0 MAX_PERIOD=2.2u' , '.options POP
```

returns 2 because the first occurrence of the literal '.options' occurs at index=2. Note that the '.' metacharacter character is escaped with the '\character' in the regular expression string.

A failing example:

```
RegExSearch( [ '.tran lm 0' , '.pop TRIG_COND=1_TO_0 MAX_PERIOD=2.2u' , '.options POP
```

returns -1 because '.ac' is not found in any of the input string array elements.

See Also

- [RegExIsValid\(\)](#)
- [RegExContains\(\)](#)
- [RegExMatch\(\)](#)
- [RegExReplace\(\)](#)
- [RegExSearchAll\(\)](#)
- [RegExStrStr\(\)](#)
- [RegExSplit\(\)](#)

4.585 RegExSearchAll

This function compiles a regular expression from the second argument and searches for a regular expression match in each element of the string array supplied as the first argument. The return value is the indexes into the input string array where the matches occur, or -1 if no matches are found. This function is a regular expression version of the [SIMPLISearchIdx\(\)](#) function.

The third argument is the offset into the input string to start the regular expression search.

In common with the other Regular Expression functions, this function is case *insensitive* by default. Also by default, the dot metacharacter character '.' does not match new line characters. You can change these behaviors using two options passed as the fourth argument.

The regular expression functions in SIMetrix/SIMPLIS support Perl-compatible regular expressions. Forward references, back references, alternation, capturing up to 99 groups, negative and positive look behind and ahead assertions are all supported. Regular expressions can be tested for validity with the [RegExIsValid\(\)](#) function.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		String array to search
2	string	Yes		Regular expression
3	real	No	0	Character offset
4	string	No		Options

Argument 1

The input string array to search for the regular expression match.

Argument 2

The regular expression string.

Argument 3

The character position to start the regular expression search. 0 starts the search at the beginning of the input string.

Argument 4

Two options are supported:

- `casesensitive`
- `dotmatchesall`

Passing the first option ‘`casesensitive`’ makes the regular expression match case sensitive. The second option ‘`dotmatchesall`’ causes the dot (‘`.`’) metacharacter to match new lines in the input string. You can pass both options as a string array:

```
[ `casesensitive' , `dotmatchesall' ]
```

Returns

Return type: real

The indexes into the input string array where the regular expression matches, or -1 indicating no matches of the regular expression are found in the input string array.

Example

A simple example where a string array is searched for a certain phrase:

```
RegexSearchAll( [ '.tran 1m 0' , '.pop TRIG_COND=1_TO_0 MAX_PERIOD=2.2u' , '.options
```

returns [2 , 3] because both the second and third indexes match the literal ‘`.options`’ string. Note that the ‘`.`’ metacharacter character is escaped with the ‘`\`’ character in the regular expression string.

A failing example:

```
RegexSearchAll( [ '.tran 1m 0' , '.pop TRIG_COND=1_TO_0 MAX_PERIOD=2.2u' , '.options
```

returns -1 because ‘`.ac`’ is not found in any of the input string array elements.

See Also

- [RegexIsValid\(\)](#)
- [RegexContains\(\)](#)
- [RegexMatch\(\)](#)
- [RegexReplace\(\)](#)
- [RegexSearch\(\)](#)
- [RegexStrStr\(\)](#)
- [RegexSplit\(\)](#)

4.586 RegExSplit

This function compiles a regular expression from the second argument and splits the first argument into a string array where the regular expression defines the delimiters. If no delimiters are found in the input string, the input string is returned unaltered. To split a string on a single character which is also escaped, see the [ScanEscape\(\)](#) and [ParseEscape\(\)](#) functions.

In common with the other Regular Expression functions, this function is case *insensitive* by default. Also by default, the dot metacharacter character '.' does not match new line characters. You can change these behaviors using two options passed as the third argument.

The regular expression functions in SIMetrix/SIMPLIS support Perl-compatible regular expressions. Forward references, back references, alternation, capturing up to 99 groups, negative and positive look behind and ahead assertions are all supported. Regular expressions can be tested for validity with the [RegExIsValid\(\)](#) function.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		String to split
2	string	Yes		Regular expression
3	string	No		Options

Argument 1

The input string to split on the regular expression match.

Argument 2

The regular expression string.

Argument 3

Two options are supported:

- casesensitive
- dotmatchesall

Passing the first option 'casesensitive' makes the regular expression match case sensitive. The second option 'dotmatchesall' causes the dot ('.') metacharacter to match new lines in the input string. You can pass both options as a string array:

```
[ 'casesensitive' , 'dotmatchesall' ]
```

Returns

Return type: string array

The input string tokenized on the regular expression.

Example

A simple example where the input string is split on a literal underscore character

```
RegExSplit( 'PARAM_MODEL_NAME' , '_' )
```

returns the array:

```
`PARAM'  
`MODEL'  
`NAME'
```

A more complex example using a negative look behind assertion. This splits the string on the underscore as long as the previous letter isn't a 'L'.

```
RegExSplit( 'PARAM_MODEL_NAME' , '(?!L)_' )
```

The return value is the two elements:

```
`PARAM'  
`MODEL_NAME'
```

An example where the regular expression is not found in the input string:

```
RegExSplit( 'PARAM_MODEL_NAME' , 'z' )
```

Returns the input string: 'PARAM_MODEL_NAME'

See Also

- [RegExIsValid\(\)](#)
- [RegExContains\(\)](#)
- [RegExMatch\(\)](#)
- [RegExReplace\(\)](#)
- [RegExSearch\(\)](#)
- [RegExSearchAll\(\)](#)
- [RegExStrStr\(\)](#)

4.587 RegExStrStr

This function compiles a regular expression from the second argument and searches for a regular expression match in the string supplied as the first argument. The return value is the index where the pattern match occurs, or an empty vector if no matches are found in the input string.

The third argument is the offset into the input string to start the regular expression search.

In common with the other Regular Expression functions, this function is case *insensitive* by default. Also by default, the dot metacharacter character '.' does not match new line characters. You can change these behaviors using two options passed as the fourth argument.

The regular expression functions in SIMetrix/SIMPLIS support Perl-compatible regular expressions. Forward references, back references, alternation, capturing up to 99 groups, negative and positive look behind and ahead assertions are all supported. Regular expressions can be tested for validity with the [RegExIsValid\(\)](#) function.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		String to search
2	string	Yes		Regular expression
3	real	No	0	Character offset
4	string	No		Options

Argument 1

The input string to find regular expression matches.

Argument 2

The regular expression string.

Argument 3

The character position to start the regular expression search. 0 starts the search at the beginning of the input string.

Argument 4

Two options are supported :

- casesensitive
- dotmatchesall

Passing the first option 'casesensitive' makes the regular expression match case sensitive. The second option 'dotmatchesall' causes the dot('.') metacharacter to match new lines in the input string. You can pass both options as a string array:

```
[ 'casesensitive' , 'dotmatchesall' ]
```

Returns

Return type: real

A real number indicating the index into the input string where the regular expression match occurs. If no match is found, returns an empty vector.

Example

Find the index where a phrase starts:

```
RegExStrStr( 'abcdef' , 'de' )
```

returns 3 because 'de' starts at the fourth character, or index=3 in the input string 'abcdef'.

Using alternation, find the location where the input string contains 'bc' or 'de':

```
RegExStrStr('abcdef', 'de|bc' )
```

returns 1 because 'bc' matches at the second character, or index=1 of the input string 'abcdef'.

A failing match:

```
RegexStrStr( `abcdef' , `^xyz' )
```

returns an empty vector. You can check the length of the return with the [Length\(\)](#) function.

See Also

- [RegexIsValid\(\)](#)
- [RegexContains\(\)](#)
- [RegexMatch\(\)](#)
- [RegexReplace\(\)](#)
- [RegexSearch\(\)](#)
- [RegexSearchAll\(\)](#)
- [RegexSplit\(\)](#)

4.588 RelativePath

Returns a path relative to the reference directory (argument 2 or current working directory) of the full path name supplied in argument 1.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Full path name
2	string	No	Current directory	Reference directory

Returns

Return type: string

See Also

- [FullPath\(\)](#)
- [SplitPath\(\)](#)

4.589 RemapDevice

Map SIMetrix simulator device to model name and level number.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Mapping spec

Argument 1

Comma delimited list of name=value pairs providing spec to map a device type to its model and level number. Name=value pairs are defined as follows:

Returns

Return type:

Notes

All device models (that is the binary code that implements the device equations) have an internal name that is used to uniquely identify it, but this name is not used externally. Instead .MODEL statements use their own name (e.g. nmos, pnp) coupled with an optional LEVEL parameter to define the actual device referred to. For example, the MOS level 3 device is referred internally as "MOS3" but the .MODEL statements use the names NMOS or PMOS and set the LEVEL parameter to 3. The mapping between NMOS and LEVEL 3 to "MOS3" is defined in an internal table which can be modified by this function.

A call to this function can add new entries to the table so providing additional methods of accessing a device. It can also modify existing entries to point to a new device.

To modify an existing mapping, you only need to provide ModelName, Device and Level values. The modelName and level must point to an existing combination that is already in use, e.g. ModelName=D and Level=1, and device would then be set to the new device that this combination is to point to, e.g. Diode3. So this is what the spec would be:

```
RemapDevice('ModelName=D,Level=1,Device=Diode3')
```

The above would make level 1 diodes use the same model as level=3. Here is another example:

```
RemapDevice('ModelName=R,Level=0,Device=HspiceRes')
```

Level=0 is the level value when the LEVEL parameter is not specified. In the case of resistors, no .MODEL statement is required at all, so the above line will change the default model used for all resistors to the Hspice model instead of the native SIMetrix model.

It is also possible to add a new mapping in which case the level and modelName parameters must be currently unused. Also when creating a new mapping the 'Letter' parameter must be specified. 'Letter' is the first letter of the component reference traditionally used to identify the type of device in SPICE netlists. For example 'Q' refers to BJTs and 'D' refers to diodes.

For example, the following entries define LEVEL=69 as a valid level for accessing the PSP 1.03 model:

```
RemapDevice('ModelName=nmos,Level=69,Device=psp103_n,report=on')
```

Note that two entries are required in order to support both n-channel and p-channel devices. The above doesn't change the existing level it adds an additional level. Both the original level number and 69 will be accepted and be equivalent.

When defining a new mapping the letter must be specified and usually this should be the letter conventionally used for the class of device. If defining a new mapping for a MOSFET, the letter 'M' should be used, for a diode the letter 'D' should be used and so on. However, the letters, 'N', 'P', 'W', 'U' and 'Y' maybe used as well for any type of device.

4.590 RemoveConfigCollection

Removes one or more entries from a configuration file collection.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Section name
2	string array	Yes		Items to remove

Argument 1

Section where entries to be removed are located

Argument 2

List of strings to remove from the collection.

Returns

Return type: real

4.591 RemoveModelFile

Uninstalls the model library paths specified in the argument.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Model path names

Returns

Return type: string

4.592 RemoveSymbolFiles

Removes a symbol file or set of symbol files from the symbol library

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		String array of symbol file paths

Returns

Return type: real

Number of library paths removed.

4.593 ResolveGraphTemplate

Evaluate template string used by graph object.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Graph object ID
2	string	Yes		Template
3	string	No		Options

Argument 1

ID of graph object whose properties are to be used in the template. See [Graph Object Identifiers - the 'ID'](#).

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](#). 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>`, `<t>` 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

Returns

Return type: string

Returns the result of evaluating the template.

Notes

This function along with [ResolveTemplate\(\)](#) are implemented using the same internal program code that implements the schematic TEMPLATE property in a netlist generation and behaves in the same way.

4.594 ResolveTemplate

Evaluate template string.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Template string
2	string	Yes		Property names
3	string	Yes		Property values
4	string	No	Input template unmodified	Return value for evaluation error

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>`, `<t>` 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.

Argument 4

If the template contains an expression enclosed in braces and the evaluation of the expression fails, the value defined in this argument is returned by the function

Returns

Return type: string

Returns the result of evaluating the template.

4.595 RestartTranDialog

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Initial stop time

Returns

Return type: real

4.596 Rms

Returns accumulative RMS value of argument

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Vector

Returns

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.

4.597 RMS1

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Vector
2	real	No	Start of input vector	Start x value
3	real	No	End of input vector	End x value

Returns

Return type: real

4.598 rnd

Returns a vector with each element a random value between 0 and the absolute value of the argument's corresponding element.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		vector

Returns

Return type: real array

Returns a random number.

4.599 RootSumOfSquares

Similar to the function [RMS1\(\)](#) but returns the root of the sum without performing an average.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Vector
2	real	No	Start of input vector	Start x value
3	real	No	End of input vector	End x value

Returns

Return type: real array

4.600 *rt*

Evaluate template string. This function is an alias to [ResolveTemplate\(\)](#)

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Template string
2	string	Yes		Property names
3	string	Yes		Property values

Returns

Return type: string

Returns the result of evaluating the template.

4.601 **SaveSpecialDialog**

Opens the dialog used by the schematic's Save Special... menu.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	No		Initial values

Returns

Return type: string array

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 '1'

4.602 Scan

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		String to scan
2	string	No		Delimiter
3	real	No		Min number of return values

Argument 1

String to scan.

Argument 2

Delimiter. Semi-colon if omitted. Only a single character is permitted. To scan with multiple delimiters, see the function [Parse\(\)](#).

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.

Returns

Return type: string array

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', '' ]
```

4.603 ScanEscape

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. Separators preceded by a backslash (\) in the input string will be escaped and the string will not split on those separators. All escaped separators are replaced by the unescaped separators in the return string array.

For example, the following call to ScanEscape():

```
ScanEscape('A,List,of \, Delimited,,Items', ',')
```

returns:

```
A
List
of , Delimited

Items
```

Note the return value has the escaped comma separator “\,” replaced with “;” and the empty token between “Delimited” and “Items” is preserved.

The default separator is semi-colon, which is the same as the [Scan\(\)](#) function. User-defined, single character separators can be supplied as the second argument. Separators are case sensitive. Unlike the [Scan\(\)](#) function, the ScanEscape function can be provided with multiple delimiters.

To demonstrate the difference between the [ParseEscape\(\)](#) and ScanEscape functions, consider the same string parsed with the [ParseEscape\(\)](#) function:

```
ParseEscape('A,List,of \, Delimited,,Items', ',')
```

returns:

```
A
List
of , Delimited
Items
```

Note that the empty token between “Delimited” and “Items” is removed by the [ParseEscape\(\)](#) function.

Like [Scan\(\)](#), the ScanEscape function can return a minimum length result by providing a integer as the third argument. For example, the following call to ScanEscape():

```
ScanEscape('A,List,of \, Delimited,,Items', ', ', 7)
```

returns two empty strings, filling out indexes 5 and 6 in the return string array:

```
A
List
of , Delimited

Items
```

This can save testing the length of the return value to determine if an optional token was provided.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		String to scan
2	String	No	semi-colon	Delimiters
3	Real	No	0	Minimum number of return values

Argument 1

String to scan

Argument 2

Delimiters. Semi-colon if omitted.

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.

Returns

Return type: string array

Returns tokens as an array of strings with the escaped delimiters replaced with unescaped delimiters. Empty fields are treated as a separate token.

See Also

- [Parse\(\)](#)
- [Scan\(\)](#)
- [ParseEscape\(\)](#)

4.604 ScriptName

Returns the full path of the currently executing script.

Arguments

No arguments

Returns

Return type: string

Returns full path of currently executing script. If the script running directly from the script editor then this function will return the path of the file in the editor if there is one. If the script editor file has never been saved then the return value will be '<LocalScript>'

4.605 Search

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		List to search
2	string	Yes		Items to search in list
3	string	No		Options

Argument 1

List to search.

Argument 2

Items to search in list.

Argument 3

Legacy option. Set to 'path' if the items being searched are file system paths. This is to enable case-sensitive searching on systems that use case-sensitive file names.

Returns

Return type: real array

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.

4.606 SearchModels

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path to search

Returns

Return type: string array

List of library specs containing model files

4.607 Seconds

Returns the number of seconds elapsed since January 1, 1970. Returned value is an integer.

Arguments

No arguments

Returns

Return type: real

4.608 Select2Dialog

Opens a dialog box with two list boxes allowing the user to select two values.

Arguments

Number	Type	Compulsory	Default	Description
1	String array	No		Initial values
2	String array	No		List entries

Argument 1

Five 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

Two 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.

Returns

Return type: string array

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.

4.609 SelectAnalysis

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.

Arguments

No arguments

Returns

Return type: real

4.610 SelectColourDialog

Opens a dialog box allowing the user to define a colour. The box is initialised with the colour specification supplied as an argument. The function returns the new colour specification.

If the user cancels the box, the function returns an empty vector.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	Specification for BLACK	Initial colour specification

Argument 1

Initial colour specification. May be the name of a colour object, an integer value as returned by [GetColourSpec\(\)](#) or a colour in the form #rrggbb

Returns

Return type: string

Colour in form #rrggbb

4.611 SelectColumns

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Input data
2	real	Yes		Field number
3	string	No	','	Delimiter

Returns

Return type: string array

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
```

4.612 SelectCount

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	'all'	Type of item to count. 'Wires', 'Instances', 'All'

Returns

Return type: real

4.613 SelectDevice

Opens parts browser dialog.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Parts data
2	string	No	No device selected	Selected device
3	string array	No		User installed models

Argument 1

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 *User's Manual/Device Library and Parts Management/Advanced Topics/Catalog Files* 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.

Argument 2

If 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.

Returns

Return type: string array

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.

4.614 SelectDialog

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Options
2	string array	Yes		List box entries

Argument 1

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

Returns

Return type: real array

The return value is the index or indexes of the items in the list box selected by the user, or 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).

4.615 SelectedProperties

Returns information about selected properties.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	'Handle'	Property name

Argument 1

Property whose value will be used to identify the instance that possesses the selected property.

Returns

Return type: string array

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 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.

4.616 SelectedStyleInfo

Returns chosen style information for the selected element. If a style name given in the argument does not exist for the selected element, then the current default style information will be returned instead.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Style names

Argument 1

The style names to return the style information for.

Returns

Return type: string array

The style information as used by that element. Each element of the array corresponds to the result for the matching input array element.

4.617 SelectedWires

Returns an array of strings holding the handles of selected wires.

Arguments

No arguments

Returns

Return type: string array

4.618 SelectFontDialog

Opens a dialog box allowing the user to define a font. The box is initialised with the font specification 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 specifications are strings that provide information about the type face, size, style and other font characteristics. Font specifications 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	Default font	Initial font specification
2	string	No		Name of object being edited

Returns

Return type: string

4.619 SelectRows

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Input data
2	string	Yes		Test string
3	string	No	0	Field number
4	string	No	';	Delimiter

Returns

Return type: string array

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
```

4.620 SelectSIMPLISAnalysis

Opens SIMPLIS choose analysis dialog box. This function reads and writes the schematic's F11 window directly.

Arguments

No arguments

Returns

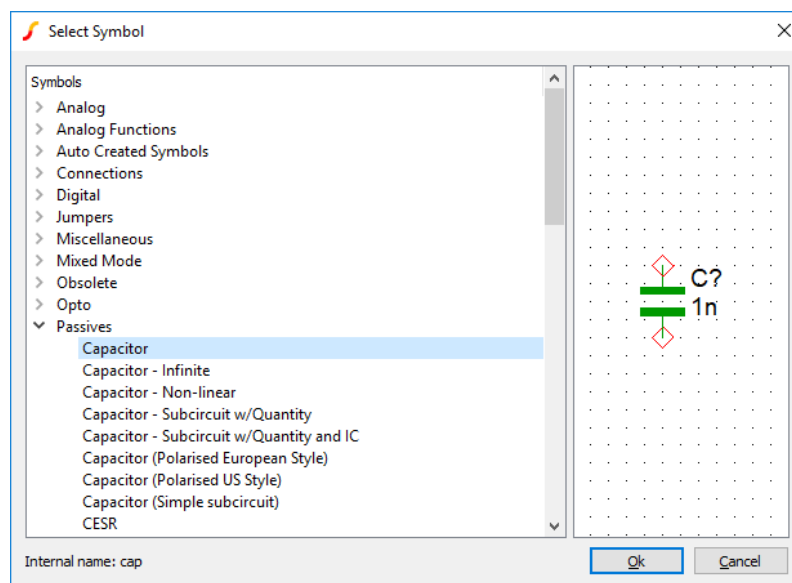
Return type: real array

The return value indicates how the user closed the box as follows, as shown in the table.

Index	Description
0	Ok pressed
1	Cancel pressed
2	Run pressed
3	No schematic open. (Dialog doesn't open in this case)

4.621 SelectSymbolDialog

Opens the following dialog box allowing the user to select a schematic symbol from the symbol library.



Arguments

Number	Type	Compulsory	Default	Description
1	string array	No	Use all installed symbols	Internal symbol names
2	string array	No	As defined by symbol	Display name and tree paths
3	string	No		Option

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.

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.

Returns

Return type: string

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.

4.622 SelGraph

Returns id of selected graph. Returns 0 if no graph is open.

Arguments

No arguments

Returns

Return type: real

Returns id of selected graph. Returns 0 if no graph is open.

4.623 SelSchem

Returns 1 if at least one schematic is open otherwise 0.

Arguments

No arguments

Returns

Return type: real

4.624 SetComponentValue

SetComponentValue is a specialised function that is used by some internal scripts. It provides a way of setting or getting a value or parameter on a schematic using a single string to identify it. This is in contrast to the usual methods to retrieve values or set values that require a sequence of commands or functions.

For example, to set a resistor R2 to 2200 ohms using conventional methods requires this sequence:

```
Unselect
Select /Prop REF R2
Prop VALUE 2200
```

With SetComponentValue, this can be done simply with:

```
Let SetComponentValue('R2', 2200)
```

However, SetComponentValue can also descend into hierarchies and set values at lower levels. For example:

```
Let SetComponentValue('U1.R2', 2200)
```

Will set the resistor R2 in hierarchical block U1.

SetComponentValue can also set named parameters. For example, if X1 is a parameterised opamp:

```
Let SetComponentValue('X1.GBW', 16.5E6)
```

will set the GBW parameter to 16.5E6.

Because the methods use to store component values and parameters is dependent on the part being edited or viewed, this function requires pre-configuring. This is done using [PrepareSetComponentValue\(\)](#). A built-in script is available that will configure SetComponentValue for the most commonly used cases. The script is called prepare_set_component_default. See [PrepareSetComponentValue\(\)](#) for further details.

Be aware that SetComponentValue will not work for all types of device - only those whose method of storing values it has been configured to accept.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Address
2	string	No	If omitted, value will not be changed but the current value will be returned	Value
3	real	No	-1	Schematic ID

Argument 3

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string array

String array of length 6 with elements defined by the following table

Index	Description
0	Current value (before being edited)
1	Status code. May be one of 'Noerr', 'BadAddress', 'AmbiguousAddress', 'IncompleteAddress', 'MissingChild', 'WriteProtected' or 'NewProp'. See table below for details
2	Full path of hierarchical schematic that contained the part that was processed
3	Handle property of instance that was processed
4	Parameter or property name that was processed
5	Debug error message. This has more detailed information than the error code

Status code	Description
'Noerr'	No error, function completed successfully
'BadAddress'	The address given was not recognised
'AmbiguousAddress'	The address given could refer to more than one item
'IncompleteAddress'	The address was incomplete. For example, it might refer to a valid part without specifying which parameter is to be written or read
'MissingChild'	Address refers to a hierarchical block which is missing, that is the schematic file could not be found
'WriteProtected'	The operation required an instance property to be edited but that property was protected and could not be edited
'NewProp'	A new property was added to the part to complete the required edit. This is not necessarily an error. Some parameters will assume default values if not present. If set to an explicit value a property may be added to the schematic instance

Notes

If the address requires a hierarchical schematic to be written, that schematic will be automatically opened.

4.625 SetDifference

Finds the difference between two sets of strings, that is the set of strings that appear in the first argument but not in the second. No string will be duplicated in the result even if duplicated in one of the inputs. This function accepts empty vectors for each argument

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		First set of strings
2	string array	Yes		Second set of strings

Argument 1

First set of strings

Argument 2

Second set of strings

Returns

Return type: string array

Difference between two sets of strings

See Also

[SetUnion\(\)](#)

[SetIntersect\(\)](#)

[SetSymmetricDifference\(\)](#)

4.626 SetInstanceParamValue

Script-based multi-step analyses use a script call to define each step. This function can be used in such a script to set an instance parameter.

Arguments

Number	Type	Compulsory	Default	Description
1	string	yes		Instance name
2	string	Yes		Parameter name
3	string	Yes		New parameter value
4	real	No	0	Vector index for vector parameters

Returns

Return type:

String indicating status of function call:

Return string	Description
'success'	Function successful
'badparam'	Unknown parameter name
'noinstance'	Unknown instance name
'nocircuit'	No circuit loaded

Example

The following script code sets the area parameter of 'Q6' to values of 100, 200 and 400 for the first, second and third steps respectively.

```
Let values = [1, 2, 4]
Let step = GetCurrentStepValue()
Let value = values[step-1]

Let SetInstanceParamValue('q6', 'area', value)
```

See Also

[GetCurrentStepValue\(\)](#)

[SetModelParamValue\(\)](#)

[GetModelParameterValues\(\)](#)

[GetDotParamValue\(\)](#)

4.627 SetIntersect

Finds the intersect of two sets of strings, that is the set of strings that appear in both arguments. No string will be duplicated in the result even if duplicated in one of the inputs. This function accepts empty vectors for each argument

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		First set of strings
2	string array	Yes		Second set of strings

Argument 1

First set of strings

Argument 2

Second set of strings

Returns

Return type: string array

Intersect set of the two arguments

See Also

[SetUnion\(\)](#)

[SetDifference\(\)](#)

[SetSymmetricDifference\(\)](#)

4.628 SetModelParamValue

Script-based multi-step analyses use a script call to define each step. This function can be used in such a script to set a model parameter.

Arguments

Number	Type	Compulsory	Default	Description
1	string	yes		Model name
2	string	Yes		Parameter name
3	string	Yes		New parameter value
4	real	No	0	Vector index for vector parameters

Returns

Return type:

String indicating status of function call:

Return string	Description
'success'	Function successful
'badparam'	Unknown parameter name
'nomodel'	Unknown model name
'nocircuit'	No circuit loaded

Example

The following script code sets the BF parameter to values of 100, 200 and 400 for the first, second and third steps respectively.

```

Let values = [100, 200, 400]
Let step = GetCurrentStepValue()
Let value = values[step-1]

```

```
Let SetModelParamValue('BC546B', 'BF', value)
```

See Also

[GetCurrentStepValue\(\)](#)

[SetInstanceParamValue\(\)](#)

[GetModelParameterValues\(\)](#)

[GetDotParamValue\(\)](#)

4.629 SetPropertyStyles

Sets whether styles are listed as property styles or not. Property styles are styles that can be applied to individual properties. This is generally meant as an internally used function only.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Styles to set as property styles
2	string array	Yes		Styles to set as not property styles

Argument 1

An array of style names that should from now on be considered as property styles.

Argument 2

An array of style names that should from now on not be considered as property styles.

Returns

Return type:

Returns nothing

4.630 SetReadOnlyStatus

Sets the read-only status of the specified schematic.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Read-only status
2	real	No	-1	Schematic ID

Argument 1

Read only status. If 1.0, will set schematic to read-only; if 0.0 will set to writeable.

Argument 2

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string

Single string defining the success of the operation is defined below.

4.631 SetSymmetricDifference

Finds the symmetric difference between two sets of strings, that is the set of strings that appear in one set but not in the other. No string will be duplicated in the result even if duplicated in one of the inputs. This function accepts empty vectors for each argument

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		First set of strings
2	string array	Yes		Second set of strings

Argument 1

First set of strings

Argument 2

Second set of strings

Returns

Return type: string array

Symmetric difference between two sets of strings

See Also

[SetUnion\(\)](#)

[SetIntersect\(\)](#)

[SetDifference\(\)](#)

4.632 SetUnion

Finds the union of two sets of strings, that is the set of strings that are in either argument. No string will be duplicated in the result even if duplicated in one of the inputs. This function accepts empty vectors for each argument

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		First set of strings
2	string array	Yes		Second set of strings

Argument 1

First set of strings

Argument 2

Second set of strings

Returns

Return type: string array

Union of the two arguments

See Also

[SetIntersect\(\)](#)

[SetDifference\(\)](#)

[SetSymmetricDifference\(\)](#)

4.633 Shell

Runs an external program and returns its exit code.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path to executable file
2	string	No		Options
3	string	No	stdout and stderr output directed to message window	File to receive redirected output

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.

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

Argument 2

String array containing one or more of the options defined in the following table:

'wait'	If specified, the function will not return until the called process has exited.
'command'	Calls OS command line interpreter to execute the command supplied. This can be used to execute system commands such as 'copy' and 'move'.
'stdout'	Stdout from the process is displayed in the command shell message window. Requires either 'wait' or file redirection see argument 3
'stderr'	Stderr from the process is displayed in the command shell message window. Requires either 'wait' or file redirection see argument 3
'console'	Opens a console window to execute the process. Disables stdout and stderr

Argument 3

If stdout or/and stderr are specified, the output can be optionally directed to a file. Use this argument to specify the file to receive the output

Returns

Return type: real array

Returns a real array of length 3 as defined below:

Index	Description
0	Process exit code. If the process is still running when this function returns, this value will be 0.
1	Error code as follows: <ul style="list-style-type: none"> 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
2	PID of process. This will be -1 if the process is no longer running

4.634 ShellExecute

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)

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File
2	string	No	none	Parameters
3	string	No	current directory	Default directory
4	string	No	'open'	Verb

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.

Argument 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.

Returns

Return type: string

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
‘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.

4.635 ShiftRef

Returns a real vector which is a copy of the original vector, but with the reference (x-axis) values shifted by the amount specified by the second argument. If the second argument is not provided, the function returns the original vector shifted so that the first data point has a reference value of 0. The y-values of the vector are unaffected by this function.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		The input vector
2	real	No		The value to shift the reference value by

Argument 1

The vector to process

Argument 2

The x axis will be shifted by this amount.

Returns

Return type: real array

The original vector, with the reference shifted by the amount specified by the second argument, or, if the second argument is not provided, the vector is shifted so the reference vector is shifted to zero.

4.636 *sign*

Returns 1 if argument is greater than 0 otherwise returns 0.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		vector

Returns

Return type: real array

Returns 1 if argument is greater than 0 otherwise returns 0.

4.637 *SimetrixFileInfo*

Returns information about a SIMetrix file. Currently this function will only return information about version 4.1 or later schematic files.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		

Argument 1

File name

Returns

Return type: string array

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)

Value	Description
'Symbol'	File is a SIMetrix component file and contains only the symbol part of the component
'SymbolSchematic'	File is a SIMetrix component file and contains both the symbol part and the schematic part of the component

4.638 SIMPLISRunStatus

Tests if a SIMPLIS simulation is running

Arguments

No arguments

Returns

Return type: string

Returns 'InProgress' if a SIMPLIS simulation is running otherwise returns 'None'.

4.639 SIMPLISSearchIdx

Searches the input string array at argument 1 for the test string passed as argument 2. Returns a real array of indices into input array where the test string matches. If no matches are found, the function returns -1. The syntax for this function is similar to the [Search\(\)](#) function, except the test string must be a single string, not an array, and the function returns all indices where the test string matches. The [Search\(\)](#) function returns only the first index where the test string matches.

The case sensitivity of the search is defined by the 3rd argument. By default the search is case insensitive. If the 3rd argument is "casesensitive", the search will only return matches using the exact case. If the third argument is omitted or any string but "casesensitive", the matches are returned for case insensitive matches against the test string. The 3rd argument itself is not case sensitive.

This function is useful for searching netlists or other tabular data for indexes where certain strings, such as control statements, are located. Typically the netlist is parsed into columns using the [SelectColumns\(\)](#) function. This selects the column where the test data is located. After the finding the indices where the data of interest is located, the original file can be edited by looping through the indices found by this function.

Arguments

Number	Type	Compulsory	Default	Description
1	String Array	Yes		List to Search
2	String	Yes		Test string
3	String	No	Empty string	Option

Argument 1

List to Search

Argument 2

The string to search the first argument for.

Argument 3

If “casesensitive” is passed, the search will be case sensitive.

Returns

Return type: real array

Array of indexes into argument 1 for the test string found in argument 2. If no matches are found the return value will be -1.

Example

A call to:

```
SIMPLISSearchIdx( [ '.INCLUDE' , 'X1' , '.Include' , 'C1' ] , '.INCLUDE' )
```

will return a vector [0 , 2]. Note the matches are by default case insensitive.

Passing the third argument as 'caseSensitive' results in a case sensitive search:

```
SIMPLISSearchIdx( [ '.INCLUDE' , 'X1' , '.Include' , 'C1' ] , '.INCLUDE' , 'caseSensitive' )
```

and will return a vector [0], indicating only the first index matches the test string.

4.640 SimulationHasErrors

Determines success of most recent simulation.

Arguments

No arguments

Returns

Return type: real

Return 1 if the most recent simulation failed with an error. Otherwise returns 0.

4.641 sin

Returns the sine of the argument specified in radians.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Returns the sine of the argument specified in radians.

4.642 *sin_deg*

Returns the sine of the argument. Result is in degrees.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Returns the sine of the argument. Result is in degrees.

4.643 *sinh*

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Return the hyperbolic sine of the argument specified in radians.

4.644 *Sleep*

Executes a timed delay.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Time in seconds

Argument 1

Delay in seconds. The function has a resolution of 100mS and so the delay will be integral multiples of that amount.

Returns

Return type: real

Function returns the value of the argument.

4.645 Sort

Performs alphanumeric sort on string array.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		String data
2	string	No		Options

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.

Returns

Return type: string array

Result is string array containing the contents of argument 1 sorted in alphanumeric order.

4.646 SortIdx

This function is deprecated. Use [SortIdx2\(\)](#) for new code.

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

Arguments

Number	Type	Compulsory	Default	Description
1	any array	Yes		Items to sort
2	string	No	'forward'	Sort direction

Argument 2

Sort option, value either 'forward' or 'reverse'. For numeric data, 'forward' returns highest values first. This is the opposite of what might be expected. [SortIdx2\(\)](#) behaves as expected and should be used in new code.

Returns

Return type: real array

An array of indexes into the input array, sorted by the method specified in argument 2.

4.647 SortIdx2

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

Use this function instead of SortIdx in new code. The original function sorted numeric values from high value to low value which is the opposite to what might usually be expected.

Arguments

Number	Type	Compulsory	Default	Description
1	any array	Yes		Items to sort
2	string	No	'forward'	Sort direction

Argument 2

Sort option, value either 'forward' or 'reverse'. 'forward' for numeric data means lowest values first.

Returns

Return type: real array

An array of indexes into the input array, sorted by the method specified in argument 2.

4.648 SourceDialog

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	<<empty>>	Initialisation string

Returns

Return type: string

4.649 SpectrumUniv

General purpose function performs a Fourier analysis on a vector. This function is used by the schematic's fixed Fourier probe.

Arguments

Number	Type	Compulsory	Default	Description
1	real vector	Yes		Input signal
2	real vector	Yes		Configuration

Argument 1

Input signal

Argument 2

real vector to configure operation on function.

Index	Description
0	Method 0: FFT

Index	Description
	1: Continuous Fourier
1	Output 0: complex 1: magnitude 2: dB 3: phase (degrees)
2	Use default resolution. Continuous Fourier only 0: Use resolution in arg3 1: Use default resolution
3	Resolution. Continuous Fourier only
4	Start frequency Actual start frequency will be $N \cdot \text{actual_resolution}$ where N is an integer
5	Stop frequency FFT: max Numpoints*resolution/2. Defaults to Numpoints*resolution/2 if $\neq 0$. Continuous Fourier: fails if $\neq 0$
6	Know fundamental frequency
7	Fundamental frequency
8	FFT only - number of points - must be 2^N . If not next value that is 2^N is used
9	FFT only - interp order
10	0: Time interval defined by arg12 and arg11 1: Use all time data
11	t start (if arg10=0)
12	t stop (if arg10=0)
13	Window 0: Rectangular 1: Hanning 2: Hamming 3: Blackman
14	Max calculation time - aborts if exceeds this value.

Returns

Return type: complex array

Fourier spectrum of input

4.650 SplitPath

Splits file system pathname into its component path.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path

Returns

Return type: string array

Return value is string array of length 4.

Index	Description
0	Drive including ':'. E.g. 'C:'
1	Directory including prefix and postfix '\'. E.g. "Program Files\SIMetrix\"
2	Filename without extension. E.g. 'SIMetrix'
3	Extension including period. E.g. '.EXE'

4.651 SplitString

Takes two values, the string and the sub string token. Returns the token removed and the string split into new sub-strings.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Input string
2	string	Yes		token

Returns

Return type: string array

String array containing the component parts of the string

Example

```
SplitString('fred/bill/jill', 'bill') ['fred/', '/jill'] SplitString('fred/bill/bill/jill', 'bill') ['fred/', '/', '/jill']
```

4.652 SprintfNumber

Returns a string formatted according to a format specification.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Format
2	real ...	No		Arguments 1-8: values

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.

Argument 2

Values used for ‘%’ format specs in the format string. Upto 8 argument values may be used.

Returns

Return type: string

Formatted string

4.653 *sqrt*

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: rea/complex array

Returns the square root of the argument.

4.654 *Str*

Returns the argument converted to a string.

Arguments

Number	Type	Compulsory	Default	Description
1	any	Yes		Input

Returns

Return type: string

The argument converted to a string.

4.655 StringLength

Returns the number of characters in the supplied string.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Input string

Returns

Return type: real

Length of the given string.

4.656 StringStartsWith

Checks whether a string starts with another string.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		String to test
2	string	Yes		String to search for

Returns

Return type: string

Returns 1 if the string starts with the given string, 0 otherwise.

4.657 StrStr

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Input string
2	string	Yes		Sub string
3	real	No	0	Offset

Argument 1

String to search

Argument 2

Sub-string

Argument 3

Offset into search string where search should begin.

Returns

Return type: real

Number of characters from start of search string where sub string starts. -1 if substring is not found.

4.658 StyleInfo

Returns the style information for the requested styles. If a requested style does not exist, the default style information is returned (unless the global flag has been set, when no data would be returned).

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Style names
2	string	No		Options flag

Argument 1

A list of style names to return the style information for. Each array element is a different style name.

Argument 2

If set to “*global*”, only global styles are returned.

Returns

Return type: string array

The style information for the requested styles. If a style does not exist and the global flag has not been set, the default style will be returned. If a style does not exist and the global flag has been set, no style information is returned for that style.

4.659 StyleLineTypes

Returns list of possible style line types.

Arguments

No arguments

Returns

Return type: string array

List of available style line types.

4.660 StyleNames

Returns a list of existing style names. If no schematic is open, only global styles will be returned

Arguments

Number	Type	Compulsory	Default	Description
1	string array	No		Optional arguments

Argument 1

If set, each element can provide an optional argument. Options are:

Argument	Description
global	Returns only global styles.
NotProperty	Returns only styles that are not property styles.
Property	Returns only styles that are property styles.

Returns

Return type: string array

List of in use style names.

4.661 SubstChar

Scans string in arg 1 and replaces characters found in arg 2 with the character specified in arg 3. This function is case sensitive. Returns the result.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		string to process
2	string	Yes		characters to replace
3	string	Yes		character to substitute

Returns

Return type: string

4.662 SubstProbeExpression

Substitutes node names in a probe expression. Used by fixed probe symbol

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Expression
2	string	Yes		Node names
3	string	Yes		Node substitutions

Argument 1

Expression to be substituted

Argument 2

Node names to be substituted

Argument 3

Node names to substitute the values in arg 1

Returns

Return type: string

Substituted expression

See Also

[ParseProbeExpression\(\)](#)

4.663 SubstString

Replaces a substring in a string. This function is case sensitive.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		String to process
2	string	Yes		Search string
3	string	Yes		Substitute string
4	string	No		Options

Argument 1

Input string.

Argument 2

Substring searched in input string. This is case sensitive when it searches.

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', then all substrings located in the string will be replaced. Otherwise, only the first occurrence will be replaced.

Returns

Return type: string

Result of string substitution. Note that only the first occurrence of the substring is replaced.

4.664 sum

Returns the sum of all values in supplied argument. If the argument is complex the result will also be complex.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Returns the sum of the supplied arguments

4.665 SumNoise

Similar to the function [RMS1\(\)](#) but returns the root of the sum without performing an average.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Vector
2	real	No	Start of input vector	Start x value
3	real	No	End of input vector	End x value

Returns

Return type: real array

Notes

This is identical to the function [RootSumOfSquares\(\)](#).

4.666 SupportedReadFormats

SIMetrix schematics and symbols can display graphical bitmap images. This function returns the formats supported.

Arguments

No arguments

Returns

Return type: string

4.667 SupportedWriteFormats

SIMetrix schematics and graphs Save Picture features can write the displayed image to a graphical file. This function returns the formats supported. See [CopyClipSchem](#) and [CopyClipGraph](#) for commands that can generate image files.

Arguments

No arguments

Returns

Return type: string array

4.668 SxUUID

Produces universally unique identifiers.

Arguments

No arguments

Returns

Return type: string

Returns a UUID string.

4.669 SymbolInfoDialog

Opens a dialog box allowing the specification of symbol details.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No		Initial settings
2	string	No		Available catalogs

Argument 1

String array length 5 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.

Returns

Return type: string array

String array of length 6 as follows

Index	Description
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'

4.670 SymbolLibraryManagerDialog

Opens the Symbol Library Manager dialog box. See *User's Manual/Symbol Editor and Library/Symbol Library Manager* for details of this feature.

Arguments

No arguments

Returns

Return type: string array

Index	Description
0	User operation: <ol style="list-style-type: none"> 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.

4.671 SymbolName

Returns symbol name of specified instance.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name
2	string	Yes		Property value
3	real	No	-1	Schematic ID

Argument 1

Along with argument 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 2

See argument 1.

Argument 3

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string

Returns the symbol name used by the instance defined by property name and value supplied in arguments 1 and 2.

4.672 SymbolNames

Returns symbol names of schematic instances.

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	-1	Schematic ID
2	string	No	Use selected	Property name
3	string	No	Use all with property name in arg 2	Property value

Argument 1

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Argument 2

Along with argument 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.

Returns

Return type: string array

String array containing the symbol names for the instances identified by this functions arguments.

Note that this function complements [PropValues2\(\)](#) and [PropFlags2\(\)](#), and will return the same number of values and in the same order as those function given the same arguments.

4.673 SymbolPinOrder

Returns pin order of symbol in currently open symbol editor sheet. Also sets new pin order if argument supplied.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	No		New pin order

Argument 1

Array of strings with names of pins in the required order.

Returns

Return type: string array

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.

4.674 SymbolPinPoints

SymbolEditor function. Returns the location of specified pin.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Pin name

Returns

Return type: real array

Returns the position of the pin as x and y values.

4.675 SystemValue

***** UNSUPPORTED ***** – See page 26 for more information

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Value name

Returns

Return type: string

4.676 SystemValuePath

***** UNSUPPORTED ***** – See page 26 for more information

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Value name

Returns

Return type: string

4.677 SystemWidgetExistsInSelectedWindow

Returns true if the system view of the type specified exists within the highlighted window.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		System view type

Argument 1

The name of the type of system view to check for. Options are:

- Command Shell
- File View
- Part Selector

Returns

Return type: boolean

True if the system view exists within the highlighted window, false otherwise.

4.678 TableDialog

Displays a spreadsheet style table to allow the user to enter tabular data. See example below for a picture.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Geometry
2	string array	No		Cell initial values

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.

Returns

Return type: string array

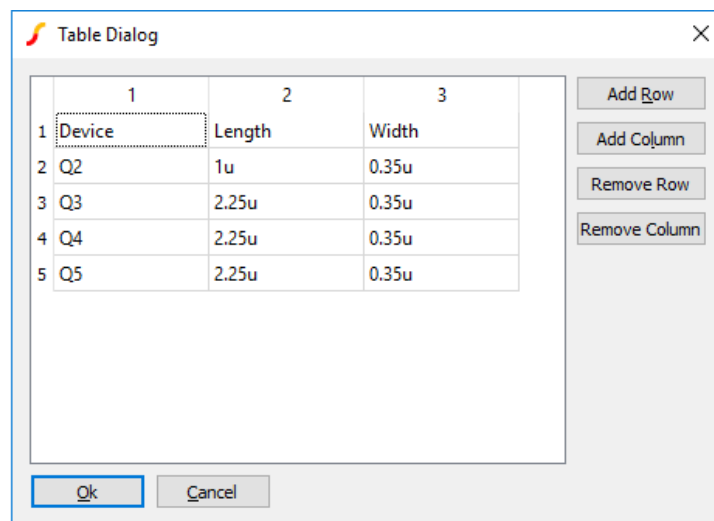
Return value will be in the same format at argument 2 and provide the contents of the cells as entered by the user.

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:



4.679 TableEditor

Displays a table of combo boxes to allow select tabular data

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Combo box entries
2	string array	Yes		Initial selection and row count
3	string array	Yes		Column, row and button labels

Argument 1

Array of strings the length of which determines the number of columns. Each entry is a '|' delimited list of strings that are used to fill the combo boxes in each cell in the corresponding column.

Argument 2

Array of strings expected to be the same length as argument 1. Specifies the initial value for the combo boxes. Can be '|' delimited in which case number of tokens determines number of rows filled for the corresponding column.

Argument 3

Length 3 array of strings providing column, row and button labels. The first element is a '|' delimited string containing the column labels. The second element is a '|' delimited string containing the row labels. The third element is 1 or 2 '|' delimited strings containing the labels for the 'Add Row' and 'Remove Row' buttons respectively.

Any or all of the elements may be empty strings in which case the default row and column labels are '1', '2', '3' etc and the button labels are 'Add Row' and 'Remove Row'.

Returns

Return type: string array

Array of strings of length equal to the number of columns. Each element is a '|' delimited string with each token holding the selected value for the corresponding row

4.680 *tan*

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

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

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

4.681 *tan_deg*

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

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

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

4.682 *tanh*

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		vector

Returns

Return type: real/complex array

Return the hyperbolic tangent of the argument.

4.683 *TemplateGetPropValue*

This function may only be used in Template scripts. These are used for advanced netlist customisation. See [Schematic Template Scripts](#) 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		REF propert value
2	string	Yes		Property name

Returns

Return type: string

4.684 TemplateResolve

This function may only be used in Template scripts. These are used for advanced netlist customisation. See [Schematic Template Scripts](#) 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		
2	string	Yes		Template value

Argument 1

REF property value

Returns

Return type: string

4.685 TextEditorHasComments

Returns whether the editor supports comments

Arguments

No arguments

Returns

Return type: boolean

4.686 ThdWeight

Returns a real array of 34 elements containing the weighting coefficients from 10Hz to 20kHz. The weighting coefficient vector can be used to calculate the weighted THD of a time-domain vector using a FFT. Note that the

The weighting coefficients are defined in the IEC 61672-1 and IEC 60537 publications. For further information, see the *IEC Website*.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		The type of weighting to return

Argument 1

Specifies the type of weighting, one of:

- “a” A-type weighting
- “b” B-type weighting
- “c” C-type weighting
- “d” D-type weighting

The weighting coefficient argument is not case sensitive.

Returns

Return type: real array

Vector of weighting coefficients with reference values from 10Hz to 20kHz.

4.687 TickCount

Returns a time in seconds since current system was started. Function may be used for timing measurement

Arguments

No arguments

Returns

Return type: real

Time in seconds since current system was started.

4.688 Time

Returns the current time in the format specified in control panel.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	<<empty>>	Options

Returns

Return type: string

4.689 ToLower

Converts a string to all lower case

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Input string

Returns

Return type: string

The input string with all characters in lower case.

4.690 ToUpper

Converts a string to all upper case

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Input string

Returns

Return type: string

The input string with all characters in upper case.

4.691 TranslateLogicalPath

Converts symbolic path to a physical path.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Symbolic path

Argument 1

Symbolic path as described in *User's Manual/Sundry Topics/Symbolic Path Names*.

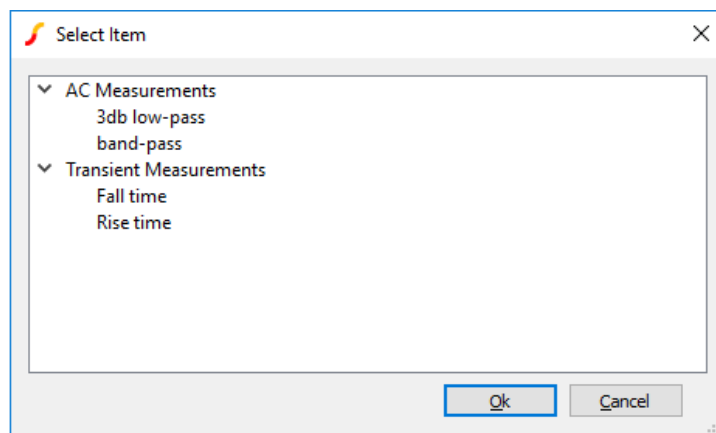
Returns

Return type: string

Returns actual file system path.

4.692 TreeListDialog

Opens the following dialog box allowing the user to specify an item in tree structured list.



Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		strings
2	string array	No	['Select Item', ',', '0', 'sort', 'false']	Options

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 unless this is set to 'nosort'
- 4 Items may selected and the box closed by double clicking unless this item is set to 'true'

Returns

Return type: real

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.

Example

The following will display the dialog box shown in the above picture:

```
Show TreeListDialog(['AC Measurements;3db low-pass', 'AC Measurements;band-pass',
+ 'Transient Measurements;Rise time', 'Transient Measurements;Fall time'])
```

4.693 True

Returns TRUE (1) if the vector specified by name in argument 1 exists AND is nonzero. 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](#) for an explanation of groups.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Vector name
2	string	No	<<empty>>	Option

Returns

Return type: real

4.694 Truncate

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		Vector
2	real	No	start of vector	Start x value
3	real	No	end of vector	end x value

Returns

Return type: real array

Example

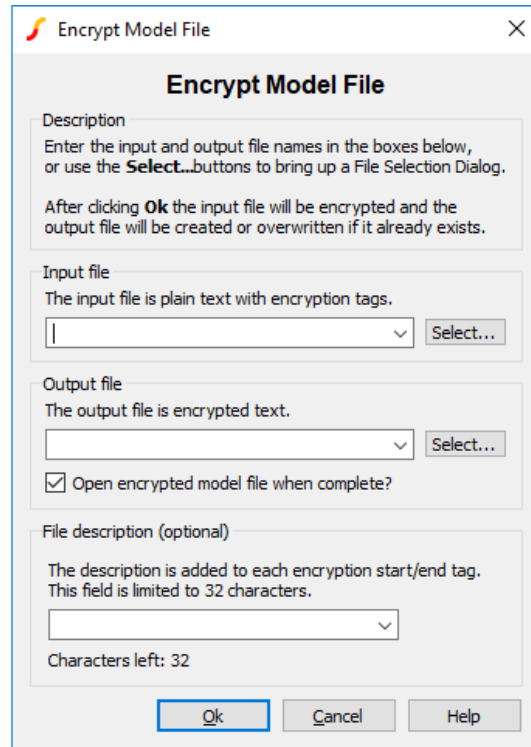
Suppose we have a vector called VOUT which was the result of a simulation running from 0 to 1mS. We want to perform some analysis on a portion of it from 250 μ S to 750 μ S. The following call to Truncate would do this:

```
Truncate(VOUT, 250u, 750u)
```

If VOUT did not actually have points at 250 μ S and 750 μ S 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.

4.695 TwoFileSelectionDialog

Opens a dialog to define two file names. While originally intended for file parsing applications, this dialog function has been made general purpose for any application where the user needs to be prompted to select two file names. The dialog has file selection buttons which open a typical File Selection Dialog. The first file is the Input file and must exist on disk when the dialog is closed. The second file is the Output file and doesn't need to exist when the dialog is closed.



TwoFileSelectionDialog Configured as the Encryption Dialog

The first argument defines the two file names and the description combo box text.

The second argument configures the displayed text on the dialog including the caption, title, group box titles and so on.

The third argument configures how the program remembers the input and output file names, description text and checkbox state. Each of these strings is a key in the user's configuration file, allowing the dialog to be used for many different applications with different memories. These remembered values will be displayed in the file and descriptive text combo boxes the next time the dialog is opened. The program remembers the last 10 file and description entries.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	No	<<empty>>	Initial files and description combo box text
2	string array	No	<<empty>>	Dialog Configuration
3	string array	No	<<empty>>	File history and other configuration

Argument 1

The argument is a string array of length 3 which defines the input file, output file and description text. Additional arguments add GUI elements to the dialog. If the 4th element is passed, the dialog is configured with a set of checkboxes inside the **Options** group at the bottom of the dialog. Each additional argument up to the 7th element represents a checkbox. The **Options** group of checkboxes was added to SIMetrix/SIMPLIS Version 8.10c.

Index	Purpose	Notes	Default
0	Input file name	populates the input file name combo box.	<<empty>>
1	Output file name	populates the output file name combo box.	<<empty>>
2	Description text	populates the description combo box.	<<empty>>
3	Checkbox state	If '1' is passed, checks the checkbox under the Output file name. If not passed or an empty string is passed, the checkbox state is determined by the memory feature.	<<empty>>
4	First Options checkbox state	If '1' is passed, checks the first checkbox in the Options Box	<<empty>>
5	Second Options checkbox state	If '1' is passed, checks the second checkbox in the Options Box	<<empty>>
6	Third Options checkbox state	If '1' is passed, checks the third checkbox in the Options Box	<<empty>>
7	Fourth Options checkbox state	If '1' is passed, checks the fourth checkbox in the Options Box	<<empty>>

Argument 2

The argument is a string array of length 13 which defines the dialog text.

Index	Purpose	Notes	Default
0	Dialog Box Caption		<<empty>>
1	Box Title		<<empty>>
2	Upper group box title		Description
3	Upper group box text		<<empty>>
4	Input file group box title		Input file
5	Input group box text		<<empty>>
6	Output file group box title		Output file
7	Output group box text		<<empty>>
8	Checkbox text		Open output file when complete?
9	Description group box title		File description (optional)
10	Description group box text		<<empty>>
11	Flag to hide the description groupbox	If set to '1' the description groupbox will be hidden.	'1'
12	Help context id	For internal use only	<<empty>>

Index	Purpose	Notes	Default
13	First Options checkbox text	Create symbol?	<<empty>>
14	Second Options checkbox text	Output debug statements to model file?	<<empty>>
15	Third Options checkbox text	Create SIMetrix compatible model?	<<empty>>
16	Fourth Options checkbox text	Add resistance tolerance to all resistors?	<<empty>>

Argument 3

The argument is a string array of length 8 which defines the memory and file selection dialog filters.

Index	Purpose	Notes	Default
0	Input file history key name	Any text without spaces, if omitted or empty string, no files will be remembered.	<<empty>>
1	Output file history key name	Any text without spaces, if omitted or empty string, no files will be remembered.	<<empty>>
2	Description history key name	Any text without spaces, if omitted or empty string, no description text will be remembered.	<<empty>>
3	Checkbox history key name	Any text without spaces, if omitted or empty string, no checkbox state will be remembered. Unlike the other memories, this only remembers the last checkbox state.	<<empty>>
4	Input file type	SIMetrix/SIMPLIS has several internally defined (and user customizable) input file types. 'Schematic' Schematic files 'Model' Model files 'Netlist' Netlist files 'Graph' Graph binary files 'Script' Script files 'VerilogA' Verilog-A files 'VerilogHDL' Verilog-HDL files 'Data' Data files 'Text' Text files 'AsciiFileEditor' Schematic ASCII Files 'LogicDef' Logic definition files used with arbitrary logic block 'Init' SIMPLIS Initialization files. An empty string will open the file browser with all files displayed.	<<empty>>

Index	Purpose	Notes	Default
5	Output file type	Same as the Input file type but for the output file extension	<<empty>>
6	Output file replacement mode	<p>'none' no replacement is performed on the output file string.</p> <p>'file' the replacement text supplied in index 7 is applied to the end of the file name before the extension. This occurs when the user selects a file using the file browser selection button.</p> <p>'ext' the replacement text supplied in index 7 is applied to the end of the file extension. This occurs when the user selects a file using the file browser selection button.</p>	
7	replacement text for index 6.		
8	First Options checkbox history key name	Any text without spaces, if omitted or empty string, no checkbox state will be remembered.	<<empty>>
9	Second Options checkbox history key name	Any text without spaces, if omitted or empty string, no checkbox state will be remembered.	<<empty>>
10	Third Options checkbox history key name	Any text without spaces, if omitted or empty string, no checkbox state will be remembered.	<<empty>>
11	Fourth Options checkbox history key name	Any text without spaces, if omitted or empty string, no checkbox state will be remembered.	<<empty>>

Returns

Return type: string array

The function returns a string array with a minimum of 4 elements. If the additional **Options** checkbox elements are passed on the first argument, the return will be of the same length as the first argument. The return is in this order:

Index	Description
0	Input file name
1	Output file name
2	Description text
3	Checkbox state
4	First Options checkbox state
5	Second Options checkbox state

Index	Description
6	Third Options checkbox state
7	Fourth Options checkbox state

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

4.696 UD

Alias of [Distribution\(\)](#)

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Tolerance
2	real array	Yes		Distribution definition

Returns

Return type: real

4.697 UngroupCurve

Ungroup a multi-step curve. Curves plotted from multi-step runs such as Monte Carlo analyses are usually grouped together as a single curve. The UngroupCurve function may be used to break the curve up into individual curves.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Curve ID

Argument 1

Curve ID to ungroup

Returns

Return type: String array

IDs of the new curves created

4.698 Unif

Returns a random number with a uniform distribution. This function is intended to be used for SIMPLIS Monte Carlo analyses and would typically be used in device value expressions.

This function is only available in the Simulator process and cannot be called from scripts running in the context of the front end. The function is only active when used by the netlist pre-processor with Monte Carlo analysis enabled. When used in other contexts, the function returns 1.0.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Tolerance

Returns

Return type: real

Random number with a uniform distribution between 1.0-tolerance and 1.0+tolerance.

Returns 1.0 when used in non Monte Carlo contexts.

Example

1k*Unif(0.1) will return 1000 +/- 10% with uniform probability distribution. Returns 1.0 in a non Monte Carlo run.

Notes

See Also

[Gauss\(\)](#)

[GaussTrunc\(\)](#)

[Distribution\(\)](#) - also alias [UD\(\)](#)

[WC\(\)](#)

[WC2\(\)](#)

4.699 Units

Returns the physical units of the argument.

Arguments

Number	Type	Compulsory	Default	Description
1	any	Yes		vector or vector name

Returns

Return type: string

Possible return values are:

- '' (meaning dimensionless)
- '?' (meaning unknown)
- 'V'
- 'A'
- 'Secs'
- 'Hertz'
- 'Ohm'
- 'Sie'
- 'F'
- 'H'
- 'J'
- 'W'
- 'C'
- 'Vs'
- 'V²'
- 'V²/Hz'
- 'V/rtHz'
- 'A²'
- 'A²/Hz'
- 'A/rtHz'
- 'V/s'

See Also

[PhysType\(\)](#)

4.700 *unitvec*

Returns a vector consisting of all 1's. Argument specifies length of vector.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		

Argument 1

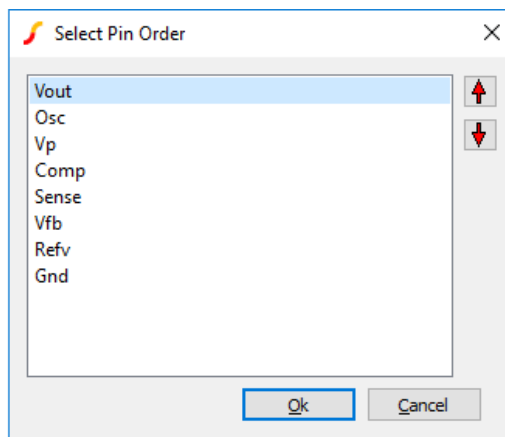
Number of elements in result

Returns

Return type: real array

4.701 UpDownDialog

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Strings to sort
2	string	No	'Select Item Order'	Box caption

Returns

Return type: string array

The strings in the new order, or an empty string if cancel is pressed.

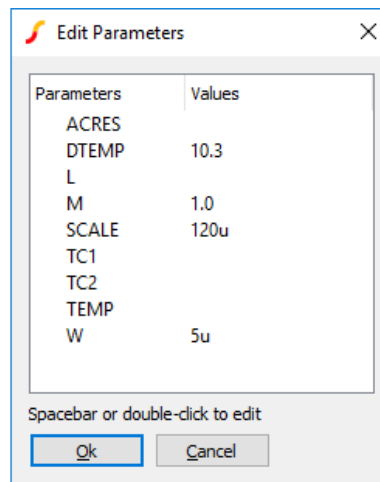
Example

The following statement will open the box as shown in the above picture

```
Show UpDownDialog(['Vout', 'Osc', 'Vp', 'Comp', 'Sense', 'Vfb', 'Refv', 'Gnd'], 'Select Pin Order')
```

4.702 UserParametersDialog

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Names
2	string	Yes		Values
3	string	No	'Edit Device Parameters'	Title

Returns

Return type: string array

Example

The following would open a dialog box as shown in the above picture:

```
Show UserParametersDialog(['ACRES', 'DTEMP', 'L', 'M', 'SCALE', 'TC1', 'TC2', 'TEMP', 'W'],
+ ['', '10.3', '', '1.0', '120u', '', '', '', '5u'])
```

4.703 Val

Returns argument converted to a value. The conversion assumes that the string supplied is an expression.

Arguments

Number	Type	Compulsory	Default	Description
1	any	Yes		Input value

Returns

Return type: real/complex

See Also

[Str\(\)](#)

4.704 ValueDialog

Opens a dialog box with up to 10 edit controls allowing numeric values to be entered.

The function returns an array representing the user selected value in each box. If cancelled it returns an empty vector.

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	1	Initial edit control values
2	string	No	<<empty>>	Edit control labels
3	string	No	<<empty>>	Dialog box caption
4	string	No	<<empty>>	

Argument 1

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.

Returns

Return type: real array

See Also

[NewValueDialog\(\)](#)

[BoolSelect\(\)](#)

[EditSelect\(\)](#)

[RadioSelect\(\)](#)

4.705 *Vec*

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Vector name
2	string	No	Current group	Group name

Returns

Return type: depends on arg 1

4.706 *vector*

Returns a vector with length specified by the argument. The value in each element of the vector equals its index.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Number of elements in result

Returns

Return type: real array

See Also

[UnitVec\(\)](#)

4.707 *VectorsInGroup*

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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	No	Current group	group name
2	string array	No		Options

Returns

Return type: string array

4.708 VersionInfo

Returns version information about running copy of SIMetrix

Arguments

No arguments

Returns

Return type: string array

Returns a string array of length 7 defined as follows:

- 0 Product name. E.g. "SIMetrix/SIMPLIS Elite with DVM"
- 1 Major Version number (3.1, 4.0 etc.)
- 2 Maintenance version. (empty or a single letter)
- 3 Internal product name. (E.g. "SIMPLIS-Elite")
- 4 Feature string allowing script to determine available functionality. This will be a combination of the following separated by the 'I' 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
- 7 Architecture : either x86 (32 bit) or x64 (64 bit). This is the architecture of the program not the operating system on which it is running
- 8 Development cycle : either Release or Beta
- 9 Build type : This will always return FinalRelease for non development versions

4.709 ViewFormattedFile

View HTML formatted text. The viewer supports basic HTML text formatting including hyperlinks to external sites.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		HTML formatted file path

Argument 1

1 or 2 element string array. Element 1 is the path to an HTML formatted file that will be displayed in a stand-alone viewer. Element 2 if supplied defines the title in the caption bar of the viewer.

Returns

Return type: string

Always returns 'ok'

4.710 ViewFormattedText

View HTML formatted text. The viewer supports basic HTML text formatting including hyperlinks to external sites.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		HTML formatted test

Argument 1

1 or 2 element string array. Element 1 is an HTML formatted text string that will be displayed in a stand-alone viewer. Element 2 if supplied defines the title in the caption bar of the viewer.

Returns

Return type: string

Always returns 'ok'

4.711 WAV_CloseFile

Close a WAV file opened with [WAV_OpenFile](#)

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		WAV file handle

Argument 1

WAV file handle obtained from [WAV_OpenFile](#)

Returns

Return type: Real

1, success, 0 fail - means that the handle is not valid

4.712 WAV_OpenFile

Open a WAV file in preparation to read its data

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		WAV File path
2	Real	No	Load all points in file	Max number of points to load

Argument 1

WAV File path

Argument 2

Maximum number of points to load

Returns

Return type: Real

4.713 WAV_ReadData

Read WAV data from file opened using [WAV_OpenFile](#)

Arguments

Number	Type	Compulsory	Default	Description
1	Real	Yes		Handle
2	Real	Yes		Options

Argument 1

Handle obtained from [WAV_OpenFile](#)

Argument 2

Options as defined below

- 0 Channel: 0 or 1
- 1 Peak value. Default = 1.0

Returns

Return type: Real array

Data

4.714 WAV_WriteFile

Write data to a file in WAV format

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Path to file
2	Real array	Yes		Configuration
3	Real	Yes		Data for channel 0 (left)
4	Real	No	None - data saved in single channel format	Data for channel 1 (right)

Argument 1

Path to file

Argument 2

Array of up to size 3 providing the configuration of the data as described in the table below

- 0 Bits per sample. Must be 8, 16, 24 or 32
- 1 Peak value. Defaults to 1.0. Input data with this value will be encoded to the maximum integer value.
- 2 Sample rate. Defaults to 44100

Argument 3

Data for channel 0 (left)

Argument 4

Data for channel 1 (right)

Returns

Return type: real

Status code:

- 0 Completed, no errors
- 1 Overload, data exceeded peak value
- 2 Format not supported
- 3 File write failed
- 4 File open failed
- 5 Exception

4.715 WC

Returns a random number with a worst case distribution. This function is intended to be used for SIMPLIS Monte Carlo analyses and would typically be used in device value expressions.

This function is only available in the Simulator process and cannot be called from scripts running in the context of the front end. The function is only active when used by the netlist pre-processor with Monte Carlo analysis enabled. When used in other contexts, the function returns 1.0.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Tolerance

Returns

Return type: real

Random number which is either 1.0+tolerance or 1.0-tolerance

Example

1k*WC(0.1) will return 900 or 1100 chosen at random. Returns 1000 in a non Monte Carlo run.

See Also

[Gauss\(\)](#)

[GaussTrunc\(\)](#)

[Distribution\(\)](#) - also alias [UD\(\)](#)

[Unif\(\)](#)

[WC2\(\)](#)

4.716 WC2

Returns a random number with a worst case distribution. This function is intended to be used for SIMPLIS Monte Carlo analyses and would typically be used in device value expressions.

This function is only available in the Simulator process and cannot be called from scripts running in the context of the front end. The function is only active when used by the netlist pre-processor with Monte Carlo analysis enabled. When used in other contexts, the function returns 1.0.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Lower tolerance
2	real	Yes		Upper tolerance

Returns

Return type: real

Random number which is either 1.0+upper-tolerance or 1.0+lower-tolerance

Example

1k*WC(0.05, -0.1) will return 900 or 1050 chosen at random. Returns 1000 in a non Monte Carlo run.

See Also

[Gauss\(\)](#)

[GaussTrunc\(\)](#)

[Distribution\(\)](#) - also alias [UD\(\)](#)

[Unif\(\)](#)

[WC\(\)](#)

4.717 WirePoints

Returns location of specified wire.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Wire handle
2	real	No	-1	Schematic ID

Argument 1

Handle of schematic wire segment. Wire handles are returned by the functions [Wires\(\)](#), [NetWires\(\)](#) and [SelectedWires\(\)](#).

Argument 2

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: real array

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 table. The functions returns an empty vector if the wire handle supplied is invalid.

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

See Also

[InstPoints\(\)](#)

4.718 Wires

Returns array of strings holding handles for all wires in the specified schematic. Wire handles are used by the function [WirePoints\(\)](#) and the commands [Select](#) and [SetHighlight](#).

Arguments

Number	Type	Compulsory	Default	Description
1	real	No	-1	Schematic ID

Argument 1

Schematic ID as returned by the function [OpenSchematic\(\)](#). 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.

Returns

Return type: string array

See Also

[NetWires\(\)](#)

[SelectedWires\(\)](#)

4.719 WM_CanRevertToSaved

Returns whether the chosen editor can be reverted to a previous saved state.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Editor ID

Argument 1

The ID of the editor to check.

Returns

Return type: boolean

Returns true (1) if the editor can be reverted to a saved state, false (0) otherwise.

4.720 WM_GetAllVisibleContentWidgetNames

Returns content windows of a given type, that are currently visible. Returns a globally unique string identifying the window instance

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		View context name

Argument 1

View context name or short name. This is a name describing the type of window. The following is a full list:

Optimiser

Symbol Editor

Ascii File Editor
 Text Editor
 Data Table
 File View
 License Editor
 Logic Definition Editor
 Netlist Editor
 Waveform Viewer
 Part Selector
 Result Analyser
 Schematic Editor
 Script Editor
 Verilog A Editor
 Verilog HDL Editor
 Web View
 Sensitivity Table

Returns

Return type: string array

Globally unique id

4.721 WM_GetCentralWidgetGeometry

Returns window geometry information for the editor region of the window.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Window name

Argument 1

Window name as returned from the function [WM_GetWindowNames](#).

Returns

Return type: string

Geometry information for the editor region.

4.722 WM_GetContentWidgetNames

Returns names of all content widgets (editors etc.) in the given window.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Window name

Argument 1

The window name, as given by the function [WM_GetWindowNames](#).

Returns

Return type: string list

Names of the content widgets within the chosen window.

4.723 WM_GetContentWidgetSessionInfo

Returns a single line string for each content widget that can be used to restore itself.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Window name

Argument 1

The window name, as given by the function [WM_GetWindowNames](#).

Returns

Return type: string array

Each element is a string that can be used to restore the widget.

4.724 WM_GetContentWidgetsLayout

Returns layout information for the content widgets (editors etc.) that can be used to restore the positioning within the window.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Window name

Argument 1

The window name, as given by the function [WM_GetWindowNames](#).

Returns

Return type: string

Layout information for the content widgets as a single string.

4.725 WM_GetContentWidgetsOfType

Returns content windows of a given type, either in a specific main window or in all windows. Returns a globally unique string identifying the window instance

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		View context name
2	String	No	Use all main windows	Main window

Argument 1

View context name or short name. This following is a full list of names that can be used:

- Optimiser
- Symbol Editor
- Ascii File Editor
- Text Editor
- Data Table
- File View
- License Editor
- Logic Definition Editor
- Netlist Editor
- Waveform Viewer or Graph
- Part Selector
- Result Analyser
- Schematic Editor or Schematic

Script Editor
 Verilog A Editor
 Verilog HDL Editor
 Web View
 Sensitivity Table

Argument 2

Main window

Returns

Return type: string array
 Globally unique id

4.726 WM_GetContentWidgetTypes

Returns the workspace view types in a particular window.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Window name

Argument 1

The window name, as stated on the window title bar, of the window to report the workspace view elements for.

Returns

Return type: string array
 List of workspace view types in the window requested.

4.727 WM_GetCurrentWidgetInfo

Returns info of most recently accessed widget (GUI window) of given type

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		Type of widget

Argument 1

Type of widget. Can be one of the following:

- Optimiser
- Symbol Editor
- Ascii File Editor
- Text Editor
- Command Shell
- Data Table
- File View
- Logic Definition Editor
- Netlist Editor
- Waveform Viewer
- Part Selector
- Result Analyser
- Schematic Editor
- Script Editor
- Verilog A Editor
- Verilog HDL Editor
- Web View
- Sensitivity Table

Returns

Return type:

8 element array providing the information shown below. The first three entries are specific to the current instance of that type. The remaining elements are general for all widgets of the given type

- 0 Full path name associated with widget
- 1 Shortened version of filename
- 2 Globally unique reference to this instance
- 3 View context name. This is the same as the argument
- 4 Main menu name. Can be used to define menus for this widget type
- 5 Short version of view context name - used to pass to event scripts
- 6 Context menu name - may be empty
- 7 tool bar name

Function returns empty if no widgets of the given type are open

See Also

[WM_GetWidgetInstanceInfo](#)

4.728 WM_GetCurrentWindowName

Returns the name of the highlighted window.

Arguments

No arguments

Returns

Return type: string

Name of the highlighted window.

4.729 WM_GetLastAccessedContentWidget

Return window name of last accessed tab sheet. This is usually the currently highlighted tab sheet.

Arguments

No arguments

Returns

Return type: string

Window name of tab sheet

4.730 WM_GetNumberModifiedEditors

Returns the number of editors that have a modified status across all windows.

Arguments

No arguments

Returns

Return type: real

Number of editors that are modified.

4.731 WM_GetPrimaryWindowName

Returns the name of the primary window.

Arguments

No arguments

Returns

Return type: string

The name of the primary window.

4.732 WM_GetSystemWidgetSessionInfo

Returns a single line string for each system widget that can be used to restore itself.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Window name

Argument 1

The window name, as given by the function [WM_GetWindowNames](#).

Returns

Return type: string array

Each element is a string that can be used to restore the widget.

4.733 WM_GetSystemWidgetsLayout

Returns layout information for the system widgets that can be used to restore the positioning within the window.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Window name

Argument 1

The window name, as given by the function [WM_GetWindowNames](#).

Returns

Return type: string

Layout information for the system widgets as a single string.

4.734 WM_GetWidgetInstanceInfo

Returns information about the specified widget (GUI Window). Widget is specified using a reference.

Arguments

Number	Type	Compulsory	Default	Description
1	String	Yes		ID of widget

Argument 1

ID of widget as returned by one of these functions

[WM_GetAllVisibleContentWidgetNames](#)

[WM_GetContentWidgetNames](#)

[WM_GetContentWidgetsOfType](#)

[WM_GetLastAccessedContentWidget](#)

Returns

Return type:

See Also

[WM_GetCurrentWidgetInfo](#)

4.735 WM_GetWindowGeometry

Returns window geometry information that can be used to restore the size and position of the chosen window.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Window name

Argument 1

The window name, as given by the function [WM_GetWindowNames](#).

Returns

Return type: string

A string representing the geometry of the window.

4.736 WM_GetWindowNames

Returns the names of all windows. This function superscedes `GetWindowNames()` which is now obsolete.

Arguments

No arguments

Returns

Return type: string array

A list of the window names. The first name will always be the primary window.

4.737 WM_NumberContentWidgets

Returns the number of content widgets in use.

Arguments

No arguments

Returns

Return type: Integer

The number of content widgets in use.

4.738 WM_NumberSystemWidgets

Returns the number of system widgets in use.

Arguments

No arguments

Returns

Return type: Integer

The number of system widgets.

4.739 WriteConfigSetting

Writes a configuration setting. Configuration settings are stored in the configuration file. See *User's Manual/Sundry Topics/Configuration Settings* 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.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Section
2	string	Yes		Key
3	string	No		Value

Argument 1

Section name

Argument 2

Key name

Argument 3

Value to set. Setting will be deleted if this is omitted.

Returns

Return type: real

See Also

[ReadConfigSetting\(\)](#)

4.740 WriteF11Lines

Writes lines directly to the F11 window overwriting any existing lines.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Lines

Argument 1

Lines to write in the form of a string array. Each element in the array creates a new line.

Returns

Return type: real

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

[ReadF11Options\(\)](#)
[WriteF11Options\(\)](#)
[GetF11Lines\(\)](#)
[AppendTextWindow](#)

4.741 WriteF11Options

Write SIMetrix simulator options to the F11 window.

Arguments

Number	Type	Compulsory	Default	Description
1	string array	Yes		Option values

Argument 1

Array of semi-colon delimited string in form:

name;value;type

name Name of option
value Value of option
type Type. One of 'BOOL', 'INT', 'REAL' or 'STRING'

The given type determines how the value is interpreted. REAL values can use engineering suffixes, e.g. 1k will be interpreted as 1000. BOOL options can have values of 'true' or '1' to indicate a true condition. All other values will be treated as false. STRING values will be entered literally.

Unlike [ReadF11Options\(\)](#), this function does not check that the option names entered are valid.

Returns

Return type:

See Also

[ReadF11Options\(\)](#)
[WriteF11Lines\(\)](#)
[GetF11Lines\(\)](#)
[AppendTextWindow](#)

4.742 WriteIniKey

Writes a value to an 'INI' file. See the function [ReadIniKey\(\)](#) for more information on INI files.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		File
2	string	Yes		Section
3	string	Yes		Key
4	string	No	Empty string	Value

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

Returns

Return type: real

Returns 1 if function successful. Otherwise returns 0.

4.743 WriteRawData

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real/complex array	Yes		data
2	string	Yes		File name
3	string	No		Options
4	string	No	'%d'	Format of index display

Returns

Return type: string

The function returns a single string according to the success or otherwise of the operation. Possible values are: 'success', 'nodata' and 'fileopenfail'.

4.744 WriteRegSetting

Writes a string value to the windows registry.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Key path
2	string	Yes		Value name
3	string	Yes		Value to be written
4	string	No	'HKCU'	Top level tree

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.

Returns

Return type: string

Returns one of three string values as defined below:

'Ok'	Function executed successfully
'WriteFailed'	Could not write that value
'InvalidTreeName'	Arg 4 invalid.

4.745 WriteSchemProp

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 the function [ReadSchemProp\(\)](#) for details.

Schematic window properties may be written to the schematic file so that they become persistent. Specify 'Save' for argument 3 to enable saving to the schematic file.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Property name
2	string	Yes		Property value
3	string array	No		Options
4	number	No		Handle

Argument 3

Legal values: 'Create', 'Save'. Note that 'Save' does not imply 'Create'. Both need to be specified to create a saveable property, i.e. ['create', 'save']

Argument 4

Handle to a schematic.

Returns

Return type: real

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

Example

To create a new persistent property:

```
Let WriteSchemProp('myproperty', 'somevalue', ['Create', 'Save'])
```

4.746 XCursor

Returns x location of graph cursor.

Arguments

No arguments

Returns

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.

4.747 XDatum

Returns x location of graph reference cursor.

Arguments

No arguments

Returns

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.

4.748 XFromY

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.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Input vector
2	real	Yes		Y value
3	real	No	2	Interpolation order (1 or 2)
4	real	No	0	Direction

Returns

Return type: real array

4.749 XMLCountElements

Returns the number of elements of a particular type at the current focus element.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Tag name
2	string	Yes		XML reference

Returns

Return type: String

The number of elements with the given tag name at the current focus level.

4.750 XMLGetAttribute

Returns the attribute value for given name at the current focus element.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		Attribute name
2	string	Yes		XML reference

Returns

Return type: String

The attribute value for the given name.

4.751 XMLGetElements

Lists elements at the current focus level.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		XML reference

Argument 1

The reference of the XML document to list the elements for.

Returns

Return type: String array

The elements that are direct children of the current focus level.

4.752 XMLGetText

Returns the text for the current focus element.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		XML reference

Returns

Return type: String

The text for the current focus element.

4.753 XMLToString

Returns the XML document as a string.

Arguments

Number	Type	Compulsory	Default	Description
1	string	Yes		XML reference

Returns

Return type: String

The XML document as a string.

4.754 XY

Creates an XY Vector from two separate vectors. An XY Vector is a vector that has a reference (see [Vector References](#)). The resulting vector will have y values defined by argument 1 and the x values (i.e. its reference) of argument 2.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		y vector
2	real array	Yes		x vector

Returns

Return type: real array

4.755 XYOrdered

Creates an XY Vector from two separate vectors. An XY Vector is a vector that has a reference (see [Vector References](#)). The resulting vector will have y values defined by argument 1 and the x values (i.e. its reference) of argument 2.

The values will be arranged in order of their x-values. This function is used to rearrange data for performance plots that are created in non-sequential order. This occurs when using multiple cores to run multi-step analyses.

Arguments

Number	Type	Compulsory	Default	Description
1	real array	Yes		y vector
2	real array	Yes		x vector

Returns

Return type: real array

4.756 YCursor

Returns y location of graph cursor.

Arguments

No arguments

Returns

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.

4.757 YDatum

Returns y location of graph reference cursor.

Arguments

No arguments

Returns

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.

4.758 YFromX

Returns array of values specifying the vertical value of the specified vector at the given x value.

Arguments

Number	Type	Compulsory	Default	Description
1	real	Yes		Input vector
2	real	Yes		X value
3	real	No	2	Interpolation order (1 or greater)

Returns

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

The following table lists all commands available

Command Name	Description
Abort	Aborts the current simulation
AbortSIMPLIS	Sends a signal to the SIMPLIS simulator instructing it to abort
About	Displays the about box
AddAnnotationText	Adds text to an annotation
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
AddDoubleClickAction	Applies a double click action to the selected elements
AddFileViewItem	Adds a FileView menu item
AddFloodFill	Symbol Definition Command. Adds a flood filled region to a symbol
AddFreeText	Adds a free text item to the currently selected graph sheet
AddGlobalStyle	Adds an additional global style.
AddImage	Adds an image to the current schematic
AddImageScript	Symbol Definition Command. Adds an image.
AddLegend	Adds a legend box to the currently selected graph
AddLegendProp	For compatibility with older versions. Creates a literal measurement object
AddPin	Symbol Definition Command. Adds a pin to a symbol
AddProp	Symbol Definition Command. Adds a property to a symbol definition

Command Name	Description
AddProperty	Adds a property to the selected schematic elements
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
AddTitleBlock	Adds a title block to a schematic
AlignText	Aligns the text of a text annotation
Anno	Annotate schematic with unique component references
AppendGroup	Appends a data group with another group
AppendTextWindow	Inserts text into the schematic editor's simulator command window (F11 Window)
Arguments	Declares arguments for a script.
BuildDefaultOptions	Resets preference settings to factory defaults
Cancel	Cancel schematic interactive command
CaptureWaveformImage	Write graph image to file in PNG format
Cd	Change current working directory
ChangeArcAttributes	Modify symbol arc attributes
ChangeSelectedStyleNames	Changes the styles of the selected elements
ChangeStyle	Changes the style of the selected elements
ChangeSymbolProperty	Modify property value/attributes in symbol editor
ClearMessageWindow	Clears the command shell message window
Close	Closes either the selected schematic or graph
CloseGraphSheet	Closes the current graph sheet
CloseLinkedRun	Close a linked simulation run.
ClosePrinter	Conclude print job
CloseSchem	Closes the current schematic
CloseSheet	Closes the currently selected schematic or symbol editor tabbed sheet
CloseSimplisStatusBox	Closes the SIMPLIS simulation status box.
CloseTextEditor	Closes the current text editor
CollectGarbage	Deletes temporary vectors
CombineMenu	Combines several menus into a separate menu
CompareSymbolLibs	Compares two symbol libraries
CompressXML	Compresses an XML file

Command Name	Description
ConvertBinaryGraph	Converts a graph binary file to XML format
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
CreateRunningDialog	Creates a dialog for displaying progress whilst a script is running
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
DecompressXML	Decompresses an XML file
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 axis.
DeleteGlobalStyle	Delete a global style.
DeleteGraphAnno	Delete graph annotation object
DeleteGraphObject	Delete graph object
DeleteShortWires	Deletes wires directly across two terminal instances
DeleteSymbolProperty	Delete property in symbol editor
DeleteToolBar	Delete a toolbar
DelGroup	Delete group (of simulation data)

Command Name	Description
DelLegendProp	Delete a graph measurement object defined by its curve and label
DelMenu	Deletes specified menu item, or submenu.
DelProp	Delete schematic instance property
DelSym	Delete symbol definition
DestroyRunningDialog	Destroys a process running dialog
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
DrawArrow	Draws an arrow in the schematic 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	Opens a text file in the text editor.
EditFont	Edit a font
EditGroupTitle	Edit a group's title
EditPaste	Paste clipboard data to schematic. (Interactive)
EditPin	Edit a pin name of a symbol in the currently installed symbol library
EndAllInteractions	Cancels all interactions occurring in all editors
EndSym	Symbol definition: terminate definition
Execute	Execute script
ExecuteMenu	Executes the menu with given identifier
FileViewCleanUpFileWatchers	Removes unnecessary file watchers
FloodFillSymbol	Flood fills a symbol
Focus	Focus on a window
FocusCommandShell	Selects the Command Shell and assigns it keyboard focus
FocusShell	Selects the Command Shell and assigns it keyboard focus
GroupSelected	Groups all selected schematic elements

Command Name	Description
Help	Display help system
HideCurve	Hides specified curve.
HighlightCurve	Highlights the selected curve
HighlightWidget	Highlights a particular content view
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 <code>/loc</code> 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
LoadModelIndex	Reload model library indexes into memory
LoadSimulatorStyleSheet	Applies a style sheet to simulator GUI elements.
LoadStyleSheet	Applies a style sheet to the whole application.
LocalDeclAll	Declare all variables in a script to be local.
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, histogram or scatter plot 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

Command Name	Description
NewAnnotation	Interactive placement of a new annotation
NewAxis	Create new y-axis
NewBasicTextEditor	Create a new plain text document
NewFileView	Creates a new File View
NewGraphWindow	Open new graph window
NewGrid	Create new graph grid
NewLabel	Adds a new unplaced text label to a schematic
NewLogicDefinitionEditor	Create a new plain text document
NewNetlist	Create a new plain text document
NewPartSelector	Creates a new Part Selector
NewPlot	Plots a curve. Uses existing graph sheet if one is available. Otherwise creates a new graph sheet.
NewPrinterPage	Start new page in print job
NewSchem	Open new schematic window.
NewScript	Create a new plain text document
NewStyle	Creates a new style
NewSymbol	Opens a new symbol editor view
NewVerilogA	Create a Verilog A editor
NewVerilogHDL	Create a Verilog HDL editor
NoPaint	Disable graph painting for duration of current script
NoUndo	Inhibits saving to undo buffer
OpenAsciiFile	Opens a schematic ASCII file in the text editor.
OpenBasicTextEditor	Opens a text file in the text editor.
OpenDirectory	Opens the directory given
OpenExternalFile	Opens the file in the operating system default
OpenGraph	a SIMetrix graph file
OpenGroup	Create new group (of simulation data) from data file.
OpenLinkedRun	Open a linked simulation run.
OpenLogicDefinitionEditor	Opens a logic definition file in the text editor.
OpenNetlist	Opens a SPICE netlist or model file in the text editor.
OpenPrinter	Begin print job
OpenRawFile	Opens a SPICE 3 format ASCII raw file.
OpenSchem	Open existing schematic
OpenScript	Opens a script source file in the text editor.

Command Name	Description
OpenSimplisStatusBox	Opens the SIMPLIS simulation status box.
OpenVerilogA	Opens a Verilog-A source file in the text editor.
OpenVerilogHDL	Opens a Verilog-HDL source file in the text editor.
OpenWebPage	Opens a web page in the system default browser
OptimiserCreate	Creates an optimiser widget (GUI) from an XML definition file
OptimiserRequestFinish	Requests that current optimisation analysis is terminated
OptionsDialog	Open options dialog box
Pan	Pan (scroll) schematic specified number of grid squares
PasteGraphImageToSchematic	Copies a picture of the graph to a schematic.
Pause	Pause current simulation
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
RedrawGraph	Redraws the chosen graph
RegisterUserFunction	Register a user defined function
RenameLibs	Run rename model utility
RenameMenu	Renames a menu item
RepeatLastMenu	Executes most recently selected the menu
ReplayTraces	Re-builds graph curves created by fixed probe definitions

Command Name	Description
Reset	Release memory used for simulation
ResizeWindow	Resizes the current window
RestartTran	Restart a transient analysis
RestoreCommandShell	Re-opens the command shell if closed
RestoreDefaultStyles	Restores default styles
Resume	Resumes a previously paused simulation.
RotInst	Rotate component or block
Run	Runs a simulation on specified netlist.
RunAsync	Spawns a new simulator process and runs specified netlist.
RunCurrentScript	Runs the script currently open in the text editor.
Save	Save selected schematic
SaveAs	Saves the currently selected schematic
SaveGraph	Saves the currently selected graph to an XML file
SaveGroup	Saves the current data group
SaveRhs	Create nodeset file to speed DC convergence
SaveSnapshot	Saves the current state of a transient analysis to a snapshot file
SaveSymbol	Save a symbol to a library or as a component
SaveSymlib	Writes the entire global symbol library or a specified installed symbol library file to <i>filename</i> . Note that the action of this command has changed significantly from that of version 4.0 and earlier.
SaveTextEditor	Saves current text editor
SaveTextEditorAs	Saves the current text editor to a specific file
SchematicEnableFileWatcher	Enables the file watcher on the schematic
SchematicFileWatcherIgnoreChanges	Disables the file watcher on a schematic editor
SchematicFileWatcherWatchChanges	Enables the file watcher on a schematic editor
ScreenShotWindow	Captures a screen shot of the current window
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

Command Name	Description
SelectLegends	Selects or unselects all graph window legends
SelectSchematic	Focuses on the specified schematic
SelectSimulator	Selects current simulator for selected schematic
SelectSymbolPin	Selects the symbol pin with given name
SelectWireConnected	Selects all wires connected to the currently selected elements
Set	Set option
SetAnnotationTextPosition	Sets the position of text within a shape based annotation
SetCurveName	Change curve name
SetDefaultEncoding	Sets the default encoding for reading text files when detected as not UTF-8.
SetDisable	Disables or re-enables schematic instances
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
SetPinPrefix	Sets the prefix for the selected pin property
SetPinSuffix	Sets the suffix for the selected pin property
SetReadOnly	Sets a vector to be read-only
SetRef	Change/attach reference to variable
SetSnapGrid	Sets schematic snap grid
SetStyleColour	Sets the style with the specified colour
SetSymbolFillStyle	Applies a fill style to a symbol
SetSymbolOriginVisibility	Controls the visibility of the origin marker in the symbol editor
SetUnits	Change physical units of variable
Shell	Execute external application or system command
ShellOld	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
SwitchModelSection	Temporarily switch model library selection
TemplateEditProperty	Template script command. Edits the property of a schematic instance

Command Name	Description
TemplateSetValue	Template script command. Sets the template value
TextEditorCommentLines	Comments highlighted lines in the selected text editor
TextEditorFileWatcherIgnoreChanges	Disables file watcher for current text editor
TextEditorFileWatcherWatchChanges	Enables file watcher for current text editor
TextEditorFind	Displays the find pop-up window for the selected text editor
TextEditorFindNext	Triggers a find next event on the current text editor
TextEditorGoToLine	Moves the cursor to the given line in the text editor
TextEditorUncommentLines	Uncomments highlighted lines in the selected text editor
TextWin	Show/hide/toggle schematic text window
Title	Change title of graph or schematic
TitleSchem	Sets the title of the current schematic
Trace	Define trace (live graphing during simulation)
UndoGraphZoom	Restore previous graph view area
UngroupSelected	Ungroups selected schematic elements
UnHighlightCurves	Unhighlights all curves.
UnLet	Delete variable
Unprotect	Unprotect and select all protected schematic instances
Unselect	Unselect all schematic instances
UnSet	Delete option
Unzip	Unzips a zip archive.
UpdateAllSymbols	Conditionally updates all symbols in open schematics
UpdateAnnotationText	Updates the text within the selected annotation
UpdateDefaultStyle	Updates the default global style.
UpdateGlobalStyle	Updates an existing global style.
UpdateGraphColours	Updates colours in all graph sheets
UpdateProperties	Restores properties on specified schematic instances to symbol defaults
UpdateRunningDialog	Updates a process running dialog by one step
UpdateStyleInfo	Updates the style information
UpdateSymbol	Updates specified symbol on selected schematic
UpdateSystemStyleInfo	Updates the style information at the system level
UpdateTitleBlock	Updates the content of a title block
UseGlobalStyles	Instruct current schematic to use global styles

Command Name	Description
ViewFile	View file in read-only mode
WebOpen	Opens a web page with the given url
Wire	Start/continue schematic wire. (Interactive unless /loc specified)
WireMode	Enter/exit schematic wiring mode
WM_CloseAllSystemWidgets	Closes all System Views in current window.
WM_CloseNonPrimaryWindows	Closes all windows except the main window
WM_CloseSheet	Closes most recently accessed content window
WM_ProgressWindowClose	Closes the specified progress window
WM_ProgressWindowCloseAll	Forces all progress windows to be closed
WM_ProgressWindowCreate	Creates a progress window
WM_ProgressWindowReport	Increments the progress bar and allows status message to be updated
WM_RevertToSaved	Reverts a widget back to its last saved state.
WM_Undock	Undocks a content widget from its window.
WriteImportedModels	Write referenced models of netlist to specified file
XMLAddAttribute	Adds an attribute to the XML at the current location
XMLAddElement	Adds an element to the XML at the current location
XMLClose	Closes the XML reference
XMLGoUpLevel	Moves the current focus element up to its parent
XMLNew	Creates a new XML reference object
XMLOpenElement	Opens the XML element and sets it as the current focus level
XMLOpenFile	Opens an XML document from a file creating a new XML reference object
XMLSaveFile	Writes an XML document to a file.
Zip	Zips a file or a directory of files
Zoom	Zoom selected schematic

5.1 Commands by Application

5.1.1 File

Cd	MCD	RD
CopyFile	MD	Unzip
Del	OpenDirectory	Zip
MakeTree	OpenExternalFile	

5.1.2 Graph

AddCurveMarker	DeleteGraphObject	Plot
AddFreeText	DelLegendProp	RedrawGraph
AddLegend	HideCurve	SaveGraph
AddLegendProp	HighlightCurve	SelectCurve
AddTextBox	MoveCurve	SelectGraph
CaptureWaveformImage	NewAxis	SelectLegends
CloseGraphSheet	NewGraphWindow	SetCurveName
CopyClipGraph	NewGrid	SetGraphAnnoProperty
CursorMode	NewPlot	ShowCurve
Curve	NoPaint	SizeGraph
CurveEditCopy	OpenGraph	Trace
DelCrv		UndoGraphZoom
DeleteAxis	PasteGraphImageToSchematic	UnHighlightCurves
DeleteGraphAnno	PlaceCursor	UpdateGraphColours

5.1.3 Lib

ListModels	MakeCatalog	SwitchModelSection
LoadModelIndex	RenameLibs	

5.1.4 Miscellaneous

About	EditColour	OpenWebPage
BuildDefaultOptions	EditFont	Quit
Close	Help	Title
CreateFont	MoveFile	ViewFile

5.1.5 Optimiser

OptimiserCreate	OptimiserRequestFinish
-----------------	------------------------

5.1.6 Printing

ClosePrinter	OpenPrinter	PrintSchematic
NewPrinterPage	PrintGraph	

5.1.7 Schematic

AddAnnotationText	Detach	SaveSymLib
AddArc	DrawArc	Select
AddCirc	DrawArrow	SelectSchematic
AddDoubleClickAction	DrawPin	SelectSimulator
AddFloodFill	EditCopy	SelectSymbolPin
AddGlobalStyle	EditCut	SelectWireConnected
AddImage	EditPaste	SetAnnotationTextPosition
AddImageScript	EditPin	SetDisable
AddPin	EndAllInteractions	SetHighlight
AddProp	EndSym	SetOrigin
AddProperty	FloodFillSymbol	SetPinPrefix
AddSeg	GroupSelected	SetPinSuffix
AddSymbolProperty	ImportSymbol	SetSnapGrid
AddTitleBlock	Inst	SetStyleColour
AlignText	MakeSymbolScript	SetSymbolFillStyle
Anno	Message	SetSymbolOriginVisibility
AppendTextWindow	Move	TemplateEditProperty
Cancel	MoveProperty	TemplateSetValue
ChangeArcAttributes	Netlist	TextWin
	NewAnnotation	TitleSchem
ChangeSelectedStyleNames	NewLabel	UngroupSelected
ChangeStyle	NewSchem	Unprotect
ChangeSymbolProperty	NewStyle	Unselect
CloseSheet	NewSymbol	UpdateAllSymbols
CompareSymbolLibs	NoUndo	UpdateAnnotationText
ConvertBinaryGraph	OpenSchem	UpdateDefaultStyle
Copy	Pan	UpdateGlobalStyle
CopyClipSchem	Probe	UpdateProperties
CopyLocalSymbol	Prop	UpdateStyleInfo
CreateSym	Protect	UpdateSymbol
Delete	RebuildSymbols	UpdateSystemStyleInfo
DeleteGlobalStyle	RestoreDefaultStyles	UpdateTitleBlock
DeleteShortWires	RotInst	UseGlobalStyles
DeleteSymbolProperty	Save	Wire
DelProp	SaveAs	WireMode
DelSym	SaveSymbol	Zoom

5.1.8 Simulator

Abort	Pause	RunAsync
AbortSIMPLIS	PreProcessNetlist	SaveRhs
CloseLinkedRun	ReadLogicCompatibility	SaveSnapShot
CloseSimplisStatusBox	Reset	ShowSimulatorWindow
Listing	RestartTran	WriteImportedModels
OpenLinkedRun	Resume	
OpenSimplisStatusBox	Run	

5.1.9 Text Editor

CloseTextEditor	OpenLogicDefinitionEditor	TextEditorCommentLines
NewBasicTextEditor	OpenNetlist	TextEditorFileWatcherIgnoreChanges
NewLogicDefinitionEditor	OpenScript	TextEditorFileWatcherWatchChanges
NewNetlist	OpenVerilogA	TextEditorFind
NewScript	OpenVerilogHDL	TextEditorFindNext
NewVerilogA	RunCurrentScript	TextEditorGoToLine
NewVerilogHDL	SaveTextEditor	TextEditorUncommentLines
OpenAsciiFile	SaveTextEditorAs	
OpenBasicTextEditor	SetDefaultEncoding	

5.1.10 User Interface

AddFileViewItem	Echo	MessageBox
Arguments	EditFile	MoveMenu
ClearMessageWindow	Execute	NewFileView
CloseSchem	ExecuteMenu	NewPartSelector
CombineMenu	FileViewCleanUpFileWatchers	OptionsDialog
CreateRunningDialog	Focus	Redirect
CreateToolBar	FocusCommandShell	RedirectMessages
CreateToolButton	FocusShell	RegisterUserFunction
DefButton	HighlightWidget	RenameMenu
DefineToolBar	Hint	RepeatLastMenu
DefKey	HourGlass	ResizeWindow
DefMenu	ListOptions	RestoreCommandShell
DeleteToolBar	LoadSimulatorStyleSheet	SchematicEnableFileWatcher
DelMenu	LoadStyleSheet	
DestroyRunningDialog		

SchematicFileWatcherIgnoreChanges	Shell	WM_ProgressWindowClose
SchematicFileWatcherWatchChanges	ShellOld	WM_ProgressWindowCloseAll
ScreenShotWindow	UnSet	WM_ProgressWindowCreate
ScriptAbort	UpdateRunningDialog	WM_ProgressWindowReport
ScriptPause	WebOpen	WM_RevertToSaved
ScriptResume	WM_CloseAllSystemWidgets	WM_Undock
ScriptStep	WM_CloseNonPrimaryWindows	
Set	WM_CloseSheet	

5.1.11 Vectors/Groups

AppendGroup	KeepGroup	SaveGroup
CollectGarbage	Let	SetGroup
CreateGroup	LocalDeclAll	SetReadOnly
DelGroup	MakeAlias	SetRef
Discard	OpenGroup	SetUnits
Display	OpenRawFile	Show
EditGroupTitle	ReplayTraces	UnLet

5.1.12 XML Processing

CompressXML	XMLClose	XMLOpenFile
DecompressXML	XMLGoUpLevel	XMLSaveFile
XMLAddAttribute	XMLNew	
XMLAddElement	XMLOpenElement	

Chapter 6

Command Reference

6.1 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.

6.2 AbortSIMPLIS

AbortSIMPLIS

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.

6.3 About

About

Displays the *about box* which provides version and copyright information.

6.4 AddAnnotationText

AddAnnotationText (text)

Adds text to the centre of the selected annotation.

Parameters

text

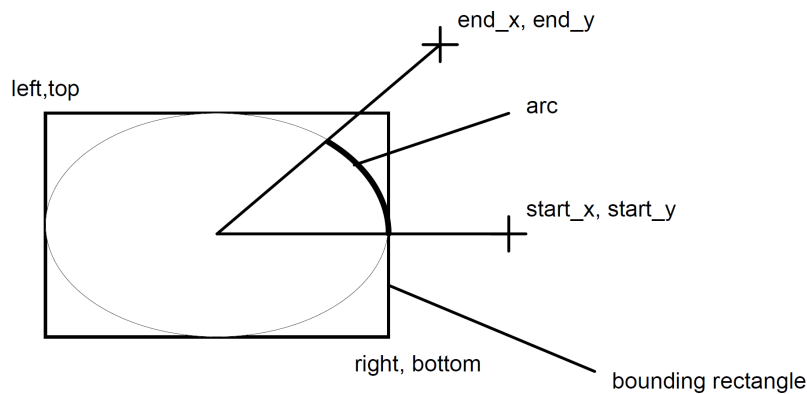
The text to add

6.5 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](#)

6.6 AddCirc

AddCirc <x-org><y-org><radius>

AddCirc is a Symbol Definition Command. Creates a circle.

Parameters

<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

6.7 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](#) for more information.

Parameters

<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 <i>y-axis</i> value that is nearest to <i>y-position</i> .
<i>label</i>	Label for marker. This may use symbolic values enclosed by '%'. See SymbolicValues for details.
<i>length</i>	Length of marker line in view units. See GraphCoordinateSystems for an explanation of view units and the view co-ordinate space. If omitted length defaults to 0.1.
<i>angle</i>	Angle of the marker line in the view co-ordinate space (See GraphCoordinateSystems). Default is 45°.

6.8 AddDoubleClickAction

AddDoubleClickAction

Adds a double click action to the selected elements. Must provide both a type and a command for the action.

Parameters

<i>/command</i>	The argument used when performing the type of action selected.						
<i>/type</i>	The type of double click action. Options are: <table> <tr> <td>OpenSchematic</td> <td>Opens the schematic defined by the command.</td> </tr> <tr> <td>OpenWebPage</td> <td>Opens the webpage defined by the command.</td> </tr> <tr> <td>RunScript</td> <td>Runs the script defined by the command.</td> </tr> </table>	OpenSchematic	Opens the schematic defined by the command.	OpenWebPage	Opens the webpage defined by the command.	RunScript	Runs the script defined by the command.
OpenSchematic	Opens the schematic defined by the command.						
OpenWebPage	Opens the webpage defined by the command.						
RunScript	Runs the script defined by the command.						

6.9 AddFileViewItem

AddFileViewItem <file-type><text><command>

Adds a menu item to a for a FileView context menu.

The functiond [GetFileViewerSelectedDirectories\(\)](#) and [GetFileViewerSelectedFiles\(\)](#) may be used to retrieve the items currently selected in the file viewer

Parameters

<i>/directory</i>	Flag to say associate this with directories.
<i>file-type</i>	The file format to associate this with, for example 'Schematic'. This corresponds to the File Extensions list in the General Options dialog. This is not required if the directory flag is set. The following values can be used: Component Schematic Script VerilogA VerilogHDL Netlist Model
<i>text</i>	The name of the menu that will be shown.
<i>command</i>	The script command to call if the menu item is executed.

See Also

[GetFileViewerSelectedDirectories\(\)](#)

[GetFileViewerSelectedFiles\(\)](#)

6.10 AddFloodFill

AddFloodFill <start-x><start-y>

AddFloodFill is a Symbol Definition Command. It is used to flood fill from the specified point in a symbol being created.

Parameters

<i>position-x</i>	Integer. Symbol x co-ordinate of position to flood fill from
<i>position-y</i>	Integer. Symbol y co-ordinate of position to flood fille from

See Also

[Schematic Symbol Script Definition](#)

6.11 AddFreeText

AddFreeText [/font <font-name>] [/colour <colour-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 [Graph Objects](#) for full details.

Parameters

<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 8 Left top 9 Centre top 10 Right top 12 Left middle 13 Centre middle 14 Right middle
<i>/colour</i>	Name of colour to be used for text. Name of option setting that will store the colour of the object in the form #rrggbb. Default is "GraphColourFreeText"
<i>/font</i>	Name of font object to be used for text object. This must with the CreateFont command. See CreateFont for details. Default is "Graph Free Text"
<i>/grid</i>	If specified, defines the grid where the text object will be placed. If omitted, text will be placed on the selected grid
<i>/localpos</i>	If set, position is stored relative to a grid. Otherwise it is relative to the whole plotting area. Typically captions are global whereas free text objects are local
<i>text</i>	The text to be displayed
<i>x-pos</i>	x-co-ordinate of the text in view units (See Graph Co-ordinate Systems). Default = 0.5.
<i>y-pos</i>	y-co-ordinate of the text in view units (See Graph Co-ordinate Systems). Default = 0.5.

6.12 AddGlobalStyle

```
AddGlobalStyle [/lineType <type>] [/lineColour <colour>] [/lineThickness <thickness>] [/fontName
<name>] [/fontSize <size>] [/fontColour <colour>] [/italics /bold 0|1] [/overline 0|1] [/underline 0|1]
[/propertyStyle 0|1] /nooverride <name>
```

Adds an additional global style to the available styles.

This will overwrite any styles with the same name unless the `nooverride` flag is set.

Parameters

<i>/bold</i>	Bold font.
<i>/fontColour</i>	As a hex value in BGR format. E.g. Blue is FF0000, Red is 0000FF, or HTML format e.g. #0000FF is Blue
<i>/fontName</i>	Font family name.
<i>/fontSize</i>	A number.
<i>/italics</i>	Italic font.
<i>/lineColour</i>	As a hex value in BGR format. E.g. Blue is FF0000, Red is 0000FF, or HTML format e.g. #0000FF is Blue
<i>/lineFixedThickness</i>	Set to 1 to fix line thickness - i.e. does not scale with zoom
<i>/lineThickness</i>	A number.
<i>/lineType</i>	Options are Solid, Dash, Dot, DashDot, DashDotDot.
<i>/nooverride</i>	Use this to ensure that the style is only added if it does not already exist as a global style.
<i>/overline</i>	Overline the text.
<i>/propertyStyle</i>	Font should appear in the Property style options drop down box.
<i>/underline</i>	Underline the text.
<i>name</i>	Name of the style to use.

6.13 AddImage

```
AddImage [/dimension <target_size>] <filename>
```

Adds an image to the current schematic. This is an interactive action and will attach the image to the cursor until it is placed on the schematic.

Parameters

<i>/dimension</i>	Target size in grid units squared. Optional.
<i>/reference</i>	If set, states that filename is actually a reference.

filename The name of the file to import as a full path or reference if the reference flag is set.

6.14 AddImageScript

AddImageScript <left><top><right><bottom><image-base64>

A Symbol Definition Command that adds an image to a symbol.

Parameters

left Integer. Left position of image.
top Integer. Top position of image.
right Integer. Right position of image.
bottom Integer. Bottom position of image.
image-base64 String. Base64 representation of the image.

6.15 AddLegend

AddLegend [/autowidth] [/font <font-name>] [/colour <colour-name>]
 [<label>[<x-pos>[<y-pos>[<width>[<height>]]]]] [/graphid id]

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](#) for more information.

Parameters

/autoWidth If specified, the width of the box will be adjusted automatically according to its contents.
/colour Name of colour to be used for text. Name of option setting that will store the colour of the object in the form #rrggb. Default is "GraphColourLegend-Box"
/font Name of font object to be used for text object. This must with the Create-Font command. See [CreateFont](#) for details. Default is "Legend Box"
/graphid Id of graph sheet. If omitted the current graph will be used

<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 Symbolic Values . The default value for label if omitted is %DefaultLabel%. This will result in the curves name and measurements being displayed in the legend box. Some alternatives are:								
	<table> <tr> <td>%Curve:Label%</td> <td>displays just the label with no measurements</td> </tr> <tr> <td>%Curve:Measurements%</td> <td>displays just the measurements</td> </tr> <tr> <td>%Curve%</td> <td>displays the curve's ID only</td> </tr> <tr> <td>%Curve:Label%/%Curve:YUnit%</td> <td>displays the curve name and y-axis units</td> </tr> </table>	%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
%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 view units (See Graph Coordinate System). 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 view units (See Graph Coordinate System). 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 = 1.02								
<i>width</i>	Physical width of box in mm. Note that this value will be ignored if /autowidth is specified. Default = 50.								
<i>height</i>	Physical height of box in mm. Default = 15mm								

6.16 AddLegendProp

AddLegendProp <curveId><property-name><property-value>

Creates a measurement object that is compatible with legend properties used in older versions of SIMetrix.

In version 8.1 and earlier, measurement values were displayed below the curve's legend and were referred to as "Legend Properties". From version 8.2, these have been replaced by Measurement objects.

Measurement objects are more advanced and have greater functionality but may perform the same function as the older legend properties.

The AddLegendProp command creates a measurement object that displays the same information as a legend property and is in all respects compatible with version 8.1 and earlier.

AddLegendProp creates a Measurement object with a Label property equal to the Name and a Template property equal to the Value. The Expression property will be empty.

This command is intended to keep old code working. New scripts should use the function [CreateGraphMeasurement\(\)](#).

Parameters

<i>curveId</i>	The curve ID. Curve id is returned by the functions GetSelectedCurves() , GetAxisCurves() and GetAllCurves() .
<i>property-name</i>	Name of property. May be any string and may contain spaces.
<i>property-value</i>	Value of property. May be any string and may contain spaces.

See Also

[CreateGraphMeasurement\(\)](#)

[EditGraphMeasurement\(\)](#)

[FindGraphMeasurement\(\)](#)

6.17 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](#) for more details.

Parameters

<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 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.

qualifier-list

One or more qualifiers used for XSPICE devices. For more information refer to *Simulator Reference Manual/Simulator Devices/Using XSPICE Devices*. 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](#)

6.18 AddProp

```
AddProp [/font <font>] [/sel] <name>[<init-value>] [<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 *User's manual/Schematic Editor/Properties*.

Parameters

<i>/font</i>	Integer from 1 - 8 that specifies one of 8 fonts as follows: <ol style="list-style-type: none"> 1 Default 2 Caption 3 Free text 4 Annotation 5 User 1 6 User 2 7 User 3 8 User 4 <p>The value specified by <i>/font fontIndex</i> overrides bits 11-13 of the <i>flags</i> value (see above).</p>												
<i>/sel</i>	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.												
<i>/styleOverride</i>	Optional override style for the property. This property will then use this style instead of the inferred default style.												
<i>name</i>	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: <table> <tr> <td>ref</td> <td>Component reference.</td> </tr> <tr> <td>value</td> <td>Component value or model name (E.g. BC547)</td> </tr> <tr> <td>model</td> <td>Single letter to signify type of device.</td> </tr> <tr> <td>netname</td> <td>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.</td> </tr> <tr> <td>schematic_path</td> <td>File system pathname for a hierarchical block.</td> </tr> <tr> <td>valuescript</td> <td>Script that is called when F7 is pressed or the menu Edit Value/Model is selected.</td> </tr> </table> <p>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.</p>	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.
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.												
<i>initvalue</i>	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’.												
<i>flags</i>	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 in the Prop command .												

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 in [Attribute Flags in the Prop command](#).

If X_pos and Y_pos are specified, the text will be displayed vertically in 90 and 270 degree rotated orientations.

Example

```
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](#)

6.19 AddProperty

```
AddProperty [/name <name>] [/value <value>]
```

Adds a property to the selected schematic elements.

Parameters

/name Name of the property.

/value Value of the property.

6.20 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.

Parameters

/overrideSelectedStyle Unsupported: Assigns a different selected style to the default Symbol style for this segment.

<i>/overrideStyle</i>	Unsupported: Assigns a different style to the default Symbol style for this segment.
<i>start-x</i>	Integer. Symbol x co-ordinate of start of segment
<i>start-y</i>	Integer. Symbol y co-ordinate of start of segment
<i>end-x</i>	Integer. Symbol x co-ordinate of end of segment
<i>end-y</i>	Integer. Symbol y co-ordinate of end of segment

See Also

[Schematic Symbol Script Definition](#)

6.21 AddSymbolProperty

AddSymbolProperty <name><flags><value>[<x><y>] [/styleoverride <style-name>]

Adds a property to the symbol currently open in the symbol editor. See the User's Manual for detailed information on properties.

Parameters

<i>/styleoverride</i>	Style name to use instead of the inferred style.
<i>name</i>	Property name
<i>flags</i>	Property attribute flags. See Attribute Flags in the Prop command .
<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

6.22 AddTextBox

AddTextBox [/font <font-name>] [/colour <colour-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.

Parameters

<i>/colour</i>	Name of colour to be used for text. Name of option setting that will store the colour of the object in the form #rrggbb. Default is GraphColourTextBox-Text
<i>/font</i>	Name of font to be used for text. This must either be a built in font or one created using CreateFont. Default is "Graph Text Box"

<i>/graphid</i>	Id of graph sheet. If omitted the current graph will be used
<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"> <i>%Date%</i> The date when the object was created <i>%Time%</i> The time when the object was created <i>%Version%</i> The name and current version of the program See Symbolic Values for more information on symbolic values.
<i>x-position</i>	The x position of the box in view units (See Graph Coordinate Systems)
<i>y-position</i>	The y position of the box in view units (See Graph Coordinate Systems)

6.23 AddTitleBlock

AddTitleBlock [*/company* <company name>] [*/title* <title name>] [*/author* <author name>] [*/loc* <x><y>] [*/notes* <notes>] [*/layout* <layout>] [*/logo* <imagedata>] [*/date* <date>] [*/version* <version>]

Adds a title block to the currently selected schematic.

Parameters

<i>/author</i>	The name of the author.
<i>/company</i>	The authoring company name.
<i>/date</i>	Optional string representing the date. If «auto» is used, this will use auto date on saving.
<i>/layout</i>	Either 'horizontal' or 'vertical'.
<i>/loc</i>	The location on the schematic to place the title block. Two integer arguments, first is x-position, second is y-position.
<i>/logo</i>	Full path to an image file to use as the logo image. Only available in the horizontal layout.
<i>/notes</i>	Notes about the schematic. Only available in the horizontal layout. The notes section can be long, however you must include a backslash n within the string to indicate where line breaks should happen in the text, otherwise the entire notes section will appear on a single line.
<i>/title</i>	The title of the schematic.
<i>/version</i>	Optional string representing the version number. If «auto» is used, this will use auto version on saving.

6.24 AlignText

AlignText

Aligns the text of a text annotation. Options are left, right or center aligned.

Parameters

<i>/center</i>	For center alignment.
<i>/left</i>	For left alignment.
<i>/right</i>	For right alignment.

6.25 Anno

Anno [/prop property_name] [/nopaint] [/bypos] [/minSuffix min_suffix]

Automatically allocates unique component references to all components on currently selected schematic.

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.

Parameters

<i>/bypos</i>	If specified, all references will be reassigned according to their position on the schematic working left to right. Unlike <i>/minSuffix</i> this switch does reassign all references. It can be used with <i>/minSuffix</i> to reassign all references in a schematic according to a desired specification.
<i>/minSuffix</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 <i>min-suffix</i> . Only values that need changing will be affected.
<i>/nopaint</i>	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.
<i>/prop</i>	If specified, annotates properties of name <i>property-name</i> . Otherwise properties of name "ref" are annotated.

6.26 AppendGroup

AppendGroup <group><appending-group>

Appends a data group with another group. Appending a group joins vectors with the same name and type in both groups to add a new division. (Refer to [Multi-division Vectors](#))

This is used for Multi-core multi-step SIMPLIS simulations. Each SIMPLIS process runs independently creating its own data file. When the processes have completed their simulations, the data files are loaded to

create groups which are then appended using this command. The end result is a multi-division vector which looks the same as if it were created by a conventional single-core run.

See Also

[CreateGroup](#)
[DelGroup](#)
[OpenGroup](#)
[Groups\(\)](#)

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

6.27 AppendTextWindow

AppendTextWindow [/file <filename>] [/text]

Inserts text into the schematic editor's simulator command window also known as the *F11 window*.

Parameters

<i>/copy</i>	If specified the text is copied to the F11 window replacing the existing text. Otherwise text is appended.
<i>/file</i>	
<i>filename</i>	If specified, the contents of the specified file is placed in the F11 window
<i>text</i>	If ' <i>/file filename</i> ' 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

[ReadF11Options\(\)](#)
[WriteF11Options\(\)](#)
[WriteF11Lines\(\)](#)
[GetF11Lines\(\)](#)

6.28 Arguments

Arguments <argument>...

6.29 BuildDefaultOptions

BuildDefaultOptions

Resets preference settings to factory defaults

6.30 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](#).

6.31 CaptureWaveformImage

CaptureWaveformImage

Copies a graphical picture of the graph to a file.

Parameters

<i>/file</i>	Specifies the filename to output to. If omitted, the image will be written to the clipboard
<i>/graphid</i>	Write image of specified graph
<i>/pixelratio</i>	Write image of specified graph
<i>/size</i>	Specifies the size of the image to capture. Values are width and height.

Notes

This command writes a graph in PNG format to a file. To write in scalable formats, use the more general [CopyClipGraph](#)

6.32 Cd

Cd [*{pathname}*]

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.

6.33 ChangeArcAttributes

ChangeArcAttributes [*{theta}*] [*{v-over-h}*]

Modifies the attributes of the selected arc or arcs in the currently open symbol editor sheet.

Parameters

<i>theta</i>	Arc swept angle in degrees. Default = 90.
<i>v-over-h</i>	vertical radius/horizontal radius. Default = 1 (i.e. a circular arc)

6.34 ChangeSelectedStyleNames

ChangeSelectedStyleNames [/noupdate] [/normal <name>] [/selected <name>]

Changes the styles of the selected elements.

Parameters

<i>/normal</i>	The name of the new normal style to use.
<i>/noupdate</i>	Do not make a visual update.
<i>/selected</i>	The name of the new selected style to use.

6.35 ChangeStyle

ChangeStyle <style-name>

Changes the style of the selected elements.

Parameters

<i>/selected</i>	If set, will change the <i>selected</i> style used.
<i>style-name</i>	The name of the style to apply.

6.36 ChangeSymbolProperty

ChangeSymbolProperty [/value <value>] [/flags <flags>] [/loc <x><y>] [/overridestyle <override-style-name>] [/prefix <prefix>] [/suffix <suffix>] [/pin] [<prop-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.

Parameters

<i>/code</i>	
<i>/flags</i>	New property attribute flags. See Attribute Flags in the Prop command
<i>/loc</i>	New absolute location. If the location was previously relative, this will be changed to absolute if this value is specified.
<i>/overridestyle</i>	Style name to use instead of the inferred style.
<i>/pin</i>	Property name specified is a pin
<i>/prefix</i>	Pin prefix for XSpice attributes
<i>/suffix</i>	Pin suffix for XSpice attributes
<i>/value</i>	New property value

See Also

[Prop](#)

[AddProp](#)

6.37 ClearMessageWindow

ClearMessageWindow

Clears the command shell message window.

6.38 Close

Close schem|graph

Closes either the selected schematic or graph, depending on argument given.

Parameters

<i>schem</i>	Use schem to close the selected schematic.
<i>graph</i>	Use graph to close all graphs.

6.39 CloseGraphSheet

CloseGraphSheet

Closes the current graph sheet. The command will complete without raising an error condition if there are no graph windows open.

6.40 CloseLinkedRun

CloseLinkedRun

Closes a linked simulation run opened using [OpenLinkedRun](#).

See Also

[OpenLinkedRun](#)

6.41 ClosePrinter

ClosePrinter

ClosePrinter is one of a number of commands and functions used for non-interactive printing. This is explained in [Non-interactive and Customised Printing](#). 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.

Parameters

`/abort` Any print job will be aborted and no print output will be produced.

See Also

[NewPrinterPage](#)

[OpenPrinter](#)

[PrintGraph](#)

[PrintSchematic](#)

[GenPrintDialog\(\)](#)

[GetPrinterInfo\(\)](#)

6.42 CloseSchem

CloseSchem

Closes the currently selected schematic.

6.43 CloseSheet

CloseSheet

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.

Parameters

/force If specified, the sheet will close unconditionally.

6.44 CloseSimplisStatusBox

CloseSimplisStatusBox

See Also

[OpenSimplisStatusBox](#)

6.45 CloseTextEditor

CloseTextEditor

Closes the currently selected text editor based widget.

Parameters

/type Optional. Specifies the type of editor to close. Options are: *LogicDefinitionEditor*, *NetlistEditor*, *ScriptEditor*, *TextEditor*, *VerilogAEditor*, *VerilogHDL**Editor*.

6.46 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.

6.47 CombineMenu

CombineMenu <menu1|menu2|...><new_menu_name>

Combines several menus into a separate menu.

Parameters

menus Set of menu names to merge, separated by '|'.

new_menu_name The name of the new menu to merge to.

6.48 CompareSymbolLibs

CompareSymbolLibs [/detail] <file1><file2>

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”.

Parameters

<i>/detail</i>	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.
<i>/difflib</i>	If specified the second library is expected to be a difference library. Symbols not found will not be reported.
<i>lib1</i>	Path of first symbol library file
<i>lib2</i>	Path of second symbol library file

6.49 CompressXML

CompressXML <read-file><write-file>

Compresses an XML file.

Parameters

<i>read-file</i>	The file to read from.
<i>write-file</i>	The file to write to.

See Also

[DecompressXML](#)

6.50 ConvertBinaryGraph

ConvertBinaryGraph <file name><out-path>

Converts a graph binary file created with version 8.1 or earlier, into the XML format used by version 8.2 and later.

Parameters

<i>File to convert</i>	File to convert
<i>File to receive XML converted result</i>	File to receive XML converted result

6.51 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.

6.52 CopyClipGraph

CopyClipGraph

Copies a graphical picture of the graph to the clipboard or to a specified file.

Parameters

<i>/file</i>	If specified, the graph is written out to the specified file in the format specified by the format switch. If not specified the graph picture is written to the clipboard. Note that the SVG format does not support output to the clipboard
<i>/format</i>	Picture format used. Choices are: <ul style="list-style-type: none"> wmf Enhanced metafile format. 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 <i>/format</i> is omitted, wmf will be used.

<i>/graphid</i>	Write image of specified graph
<i>/mark</i>	If specified, markers are displayed on each curve as a means of identification. This is enabled automatically if <i>/mono</i> is specified.
<i>/mono</i>	Copy graph in monochromatic form.
<i>/pixelratio</i>	Write image of specified graph
<i>/transparent</i>	Image is transparent
<i>/vp</i>	Viewport dimensions in pixels. This is used to specify the size of the image if a bitmapped format (png, jpeg, bmp) is specified. x is the width, y 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.

6.53 CopyClipSchem

CopyClipSchem

Parameters

<i>/file</i>	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.
<i>/format</i>	Picture format used. Choices are: <ul style="list-style-type: none"> 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 <i>/format</i> is omitted, wmf will be used.
<i>/mono</i>	Copy schematic in monochromatic form.
<i>/vp</i>	Viewport dimensions in pixels. This is used to specify the size of the image if a bitmapped format (png, jpeg, bmp) is specified. x is the width, y 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.

6.54 CopyFile

CopyFile [/force] <from-file><to-file>

Copies a file.

Parameters

<i>/force</i>	If specified, <i>to-file</i> will be overwritten if it exists. Otherwise if <i>to-file</i> exists, the command will fail.
<i>from-file</i>	Source file
<i>to-file</i>	Destination file

6.55 CopyLocalSymbol

CopyLocalSymbol <symbol-name>[<new-symbol-name

Copies a symbol in the currently selected schematic to the global library.

Parameters

<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

6.56 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.

Parameters

<i>font-name</i>	Name of new font
------------------	------------------

font-base 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, Standard-Medium or StandardLarge.

6.57 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](#).

Parameters

/title 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\('title'\)](#)

label Base name of group. The actual group name will be appended by a number to make it unique. The new group will become the current group. To find the name actually used, you can call the function [Groups\(\)](#) immediately after calling this command. The first element of [Groups](#) (i.e. `Groups()[0]`) is always the current group.

See Also

[DelGroup](#)

[OpenGroup](#)

[Groups\(\)](#)

6.58 CreateRunningDialog

CreateRunningDialog

Creates a dialog for displaying progress whilst a script is running.

Parameters

/abortcommand Command to be executed when Abort accepted. Typically this would be a command to assign a global variable which the running script would test. For example, 'Let global:abortScript=1'. The script would test this value at appropriate times then exit cleanly. If this switch is omitted, the script will execute the [ScriptAbort](#) command which will abort the execution of the script immediately.

/abortmessage Message shown when pressing abort

/caption Title bar caption.

/displaymessage The message displayed inside the dialog

<i>/status</i>	Initial status message.
<i>/steps</i>	The number of progress steps that will occur. If 0 or not set, no progress bar will be shown.

See Also

[UpdateRunningDialog](#)

[DestroyRunningDialog](#)

6.59 CreateSym

CreateSym [/local] [/file <libfile>] [/flags <flags>] <symbol-name>[<description>][<catalog>]]

Parameters

<i>/file</i>	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.sxslb in the Symbol-Libs directory.
<i>/flags</i>	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.
<i>/local</i>	If specified, the symbol will be created in the currently open schematic and will not be saved to the global library.
<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 function GetSymbols()). This is known as the “user name” in the user interface.
<i>catalogue</i>	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”.

6.60 CreateToolBar

CreateToolBar <context-name><toolbar-name>

Creates a new empty toolbar. To add buttons to the toolbar use command [DefineToolBar](#).

Parameters

<i>/vert</i>	Toolbar will be positioned on a new row below the standard toolbar. Otherwise it will be placed to right of the standard toolbar	
<i>context-name</i>	Name of window where toolbar is to reside. Must be one of:	
	CommandShell	Command shell window
	Schematic	Schematic windows
	SimetrixSchemToolBar	Schematic windows (SIMetrix mode only)
	SimplisSchemToolBar	Schematic windows (SIMPLIS mode only)
	Symbol	Symbol editor windows
	Graph	Graph windows
	AsciiFileEditor	ASCII Schematic file editor windows
	LogicDefinitionEditor	Logic definition editors
	NetlistEditor	Netlist and model text editors
	ScriptEditor	Script text editors
	TextEditor	Plain text editors
	VerilogAEditor	Verilog-A editors
	VerilogHDLEditor	Verilog-HDL editors
	WebView	Web view windows

toolbar-name 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 are:

AsciiFileEditor	ASCII Schematic file editor windows
Shell	Command Shell toolbar
LogicDefinitionEditor	Logic definition editors
NetlistEditor	Netlist and model text editors
SimetrixSchemToolBar	Schematic windows (SIMetrix mode only)
SimplisSchemToolBar	Schematic windows (SIMPLIS mode only)
ScriptEditor	Script text editors
SymbolMain	Symbol editor toolbar
TextEditor	Plain text editors
VerilogAEditor	Verilog-A editors
VerilogHDLEditor	Verilog-HDL editors
GraphMain	Graph window toolbar
WebView	Web view windows

This name is used to reference the tool bar in the [DefineToolBar](#) command.

Notes

It is legal to define multiple toolbars with the same name but a different context name. This allows the same toolbar to be placed in multiple contexts. For example, the following is allowed:

```
CreateToolBar VerilogAEditor verilogtoolbar
CreateToolBar VerilogHDLEditor verilogtoolbar
```

The toolbar will show in both Verilog-A and Verilog-HDL editors

Changes to the toolbar configuration of a window type (i.e. context), do not take effect until all windows of that type have been closed.

See Also

[CreateToolBarButton](#)

[DefButton](#)

[GetToolBarButtons\(\)](#)

6.61 CreateToolBarButton

```
CreateToolBarButton [/toggle] [/shortcut key] [/class class-name] <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](#).

Parameters

<i>/class</i>	This is used with the function GetToolButtons() 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>/shortcut</i>	Specifies a shortcut key that will perform the same action as the tool button. For key codes see DefKey .
<i>/toggle</i>	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>name</i>	Name of button. This may be one of the pre-defined types described in DefineToolBar 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 or you may use a user defined image specified in a file. The file must be located at simetrix-root/support/images. where simetrix-root 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.
<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](#)

[GetToolButtons\(\)](#)

6.62 CursorMode

CursorMode onloffltogglelsteplstepreflstepshiftlsteprefshift

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.

Parameters

<i>on</i>	Switch cursors on
<i>off</i>	Switch cursors off
<i>toggle</i>	Toggles onloff
<i>step</i>	Step cursor to next curve
<i>stepref</i>	Step reference cursor to next curve

<i>stepShift</i>	Steps cursor to next curve within a group. Curves are grouped - for example - for Monte Carlo runs.
<i>stepRefShift</i>	Steps reference cursor to next curve within a group. Curves are grouped - for example - for Monte Carlo runs.

6.63 Curve

Curve [/xl <xlimit_low><xlimit_high>] [/yl <ylimit_low><ylimit_high>] [/xdelta <xdelta>] [/ydelta <ydelta>] [/ylabel <ylabel>] [/xlabel <xlabel>] [/yunit <yunit>] [/xunit <xunit>] [/title] [/xauto] [/yauto] [/xlog] [/ylog] [/loglog] [/dig] [/select] [/newaxis] [/newgrid] [/axisid <id>] [/autoaxis] [/coll] [/name] [/bus <bus-spec>] [/icb <objid>] [/new] [/newsheet] [/autoxlog] [/autoylog] [/histo] <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.

Parameters

<i>/autoaxis</i>	
<i>/autoXlog</i>	
<i>/autoYlog</i>	
<i>/axisid</i>	If specified, the new curve will be added to a y-axis with the id specified. Axis id is returned by the functions GetAllYAxes() , GetCurveAxis() and GetSelectedYAxis() .
<i>/bus</i>	If specified, the new curve will be plotted on a digital axis and will be plotted as a bus curve. type may be 'hex', 'dec' or 'bin' specifying hexadecimal, decimal or binary display respectively.
<i>/cv</i>	
<i>/dig</i>	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.
<i>/gridid</i>	If specified, the curve will be plotted on the specified grid.
<i>/group</i>	
<i>/guid</i>	
<i>/histo</i>	Plot histogram
<i>/icb</i>	Specifies the internal clipboard as the source of the curve data. clipboard-index is a value of 0 or more that indicates which curve in the internal clipboard is to be used. The function HaveInternalClipboardData() may be used to determine the number of curves available. The maximum acceptable value for clipboard-index is thus one less than the value returned by HaveInternalClipboardData() .
<i>/loglog</i>	Only effective when graph sheet is empty. Forces both y and x axes to be logarithmic
<i>/name</i>	If specified, curve will be named curve-name.

<i>/new</i>	
<i>/newAxis</i>	If specified, the new curve will be plotted on a new y-axis.
<i>/newGrid</i>	If specified, the new curve will be plotted on a new grid.
<i>/newGridDetached</i>	
<i>/scatter</i>	Plot scatter plot
<i>/select</i>	If specified, the new curve will be selected.
<i>/title</i>	Plot and NewPlot commands only: sets graph window title when new sheet created
<i>/ungroup</i>	
<i>/xauto</i>	Flag. Use automatic limits for x-axis. If this appears after a <i>/xl</i> specification <i>/xauto</i> will override it and vice-versa.
<i>/xAxisid</i>	If specified, the new curve will be added to a x-axis with the id specified.
<i>/xdelta</i>	Specify spacing between major grid lines on x-axis. Followed by <i>x-grid-spacing</i> , a real value. For default spacing use 0.
<i>/xl</i>	Use fixed limit for x-axis. Followed by <i>x-low-limit</i> and <i>x-high-limit</i> , which are real valued lower and upper limit of the x-axis.
<i>/xlabel</i>	Specify a label for the x-axis. If the label name argument contains any spaces, the whole string must be enclosed in double quotes.
<i>/xlog</i>	Only effective when graph sheet is empty. Forces logarithmic xaxis.
<i>/xunit</i>	Specify units for the x-axis (Volts, Watts etc.), followed by the unit name as a string. If the string contains spaces, the whole string must be enclosed in quotes (""). You should not include an engineering prefix (m, K etc.). This switch assigns the unit to the axis and also to the curve being plotted.
<i>/yauto</i>	Flag. Use automatic limits for y-axis. If this appears after a <i>/yl</i> specification <i>/yauto</i> 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> , a real value. For default spacing use 0.
<i>/yl</i>	Use fixed limit for y-axis. Followed by <i>y-low-limit</i> and <i>y-high-limit</i> , which are real valued lower and upper limit of the y-axis.
<i>/ylabel</i>	Specify a label for the y-axis. If the label name argument contains any spaces, the whole string must be enclosed in double quotes.
<i>/ylog</i>	Only effective when graph sheet is empty. Forces logarithmic yaxis.
<i>/yunit</i>	Specify units for the y-axis (Volts, Watts etc.), followed by the unit name as a string. If the string contains spaces, the whole string must be enclosed in quotes (""). You should not include an engineering prefix (m, K etc.). This switch assigns the unit to the axis and also to the curve being plotted.

Notes

***/autoxlog* and */autoylog* log test**

The x-values are deemed to be logarithmically spaced if three values satisfy the following:

$$1.0000001 > \frac{x_1^2}{x_0 * x_2} > 0.9999999$$

Where:

$$x_0 = x[0]$$

i.e the first point in the data.

If there are an even number of points:

$$x_1 = x[\frac{n}{2} - 1]$$

$$x_2 = x[n - 2]$$

where n is the number of points in the data.

If there are an odd number of points:

$$x_1 = x[\frac{n - 1}{2}]$$

$$x_2 = x[n - 1]$$

where n is the number of points in the data.

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.

6.64 CurveEditCopy

CurveEditCopy <curve-id>[<curve-id>...]

Copy specified curves to the internal clipboard. Curves so copied may be subsequently plotted using the command [Curve](#) with the /icb switch.

Parameters

<i>curve-id</i>	Id of curve. A number of functions return this value including GetSelectedCurves() .
-----------------	--

See Also

[Curve](#)

[HaveInternalClipboardData\(\)](#)

6.65 DecompressXML

DecompressXML <read-file><write-file>

Decompresses an XML file that has been compressed by SIMetrix.

Parameters

<i>read-file</i>	The file to read from.
<i>write-file</i>	The file to write to.

See Also

[CompressXML](#)

6.66 DefButton

```
DefButton [/immediate] [/comgroup <command-group>] <button-name><command>[<upCommand>]
          [/menu <menu-item-title>] [/features <features-required-for-menu-item>]
```

Defines the command executed when a button is pressed.

Parameters

<i>/comgroup</i>	This can be used with the function GetLastCommand() . GetLastCommand returns the text of the most recent command executed which specified the supplied command group value. The command DefMenu also uses this feature.
<i>/immediate</i>	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 the command DefMenu . You may not call a script if immediate execution is specified.
<i>/menu</i>	Flags whether this entry is a submenu of the button.
<i>button-name</i>	Name of button. Either a pre-defined button as listed in the command DefineToolBar or a new button created with CreateToolButton .
<i>command</i>	Command to be executed when the button is pressed. If <i>/immediate</i> is <i>not</i> specified this may be any valid command including a script.
<i>up-command</i>	Command to be executed when a toggle button is released. The button must be defined to have a toggle action using the <i>/toggle</i> switch for the command CreateToolButton .

See Also

[GetToolButtons\(\)](#)

6.67 DefineToolBar

DefineToolBar <toolbar-name><button-defs>

Defines the buttons for a user defined toolbar created using the command [CreateToolBar](#). To define the buttons for a pre-defined toolbar, the associated option setting must be set using the command [Set](#).

Parameters

<i>/reserveundefined</i>	Undefined buttons are stored and added to the toolbar automatically if defined at a later time
<i>toolbar-name</i>	Name of toolbar. This must be a toolbar created using CreateToolBar .
<i>button-defs</i>	Semi-colon delimited list of button names to add to the toolbar. Buttons may either be one defined using CreateToolButton or one of the pre-defined types shown in the table below. The '-' character may also be used to specify a spacer For a list of buttons, see Pre-defined Buttons . 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](#)

[GetToolButtons\(\)](#)

6.68 DefKey

DefKey <key-label>[<command-string>][<options>]]

DefKey is used to define custom key strokes.

Parameters

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. Valid key labels are:

F1	F2	F3	F4	F5	F6
F7	F8	F9	F10	F11	F12
INS	DEL	HOME	END	PGUP	PGDN
LEFT	RIGHT	UP	DOWN	TAB	BACK
ESC	NUM1	NUM2	NUM4	NUM5	NUM6
NUM7	NUM8	NUM9	NUM0	NUM*	NUM/
NUM+	NUM-	NUM.			

Additionally, all letter and number keys, referred to by letter/number alone. The space bar can be used (_SPACE), but must always be shifted. Shifted keys are keys that have shift, control or alt also pressed at the same time. Any of the above keys can be 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.

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.

The following commands can be used with the flag set to immediate mode:

[Cancel](#)
[DefMenu](#)
 DefKey
[Echo](#)
[Let](#)
[Move](#)
[Pan](#)
[Pause](#)
[Quit](#)
[RotInst](#)
[Select](#)
[ScriptAbort](#)
[ScriptPause](#)
[ScriptResume](#)
[Shell](#)
[Wire](#)
[Zoom](#)

Note, the command [Let](#) 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.

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](#).

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 | Options | Edit Startup Script** and add the line above.

Example

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.

6.69 DefMenu

```
DefMenu [/immediate] [/shortcut] [/norepeat] [/id <command-id>] [/comgroup <command-group>]
<menu-path>[ <command-string>[ <when-enabled>]]
```

Defines custom menu. Supersedes DefItem.

Parameters

<i>/comgroup</i>	This can be used with the function GetLastCommand() . GetLastCommand returns the text of the most recent command executed which specified the supplied command group value. The command DefButton also uses this feature.
<i>/forceUpdateCommand</i>	If set, this will force any update that occurs to also update the command, even if the command is an empty string.
<i>/forceUpdateExpression</i>	If set, this will force any update that occurs to also update the expression, even if the expression is an empty string.
<i>/id</i>	This item is used by the edit menu GUI. It is not needed for regular use.
<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>/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>/pos</i>	Position of menu. ‘1’ means the top position. If omitted, the menu is placed at the bottom. Position must also take into account any link breaks within a menu.

/shortcut 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](#), 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.

menuname Composed of strings separated by pipe symbol: '|'. First name must be one of the following:

AsciiFileEditor Schematic ASCII file text editor
 GraphMain Graph main menu
 LogicDefinitionEditor Logic definition file text editor
 NetlistEditor Netlist/Model file text editor
 ScriptEditor Script file text editor
 Shell Command shell menu
 SimetrixMain Schematic main menu - SIMetrix mode
 SimplisMain Schematic main menu - SIMPLIS mode
 SymbolMain Symbol editor fixed menu
 TextEditor Basic text editor
 VerilogAEditor Verilog-A file text editor
 VerilogHDLEditor Verilog-HDL file text editor
 WebView Web browser
 Graph Graph context menu
 Simetrix Schematic context menu SIMetrix mode
 Simplis Schematic context menu - SIMPLIS mode
 Symbol Symbol editor context menu

The menuname for fixed menus 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.

The menuname for context menus 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.

Names defined using the CombineMenu command may also be used. The names SchemMain and Schem are defined in the standard startup script using [CombineMenu](#) and provide compatibility with version 7.2 and earlier See examples below.

when-enabled

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.

Expressions enclosed in curly braces may also be used. Such expressions may contain any script expression to make customised menu enables. Care should be taken when using this feature and it should be used sparingly. Expressions can take a long time to evaluate and this will lead to sluggish menu activation response.

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 not case-sensitive, so if an existing menu exists 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.

Example

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"
```

6.70 Del

Del [/noerror] filespec

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.

Parameters

/noerror

6.71 DelCrv

DelCrv <curve-id>|<curve-name>[...]

Deletes the specified curve or curves on the selected graph. *curve_id* is returned by the functions [GetSelectedCurves\(\)](#), [GetAxisCurves\(\)](#) and [GetAllCurves\(\)](#).

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\(\)](#).

6.72 Delete

Delete

Deletes the currently selected components and/or wires in the selected schematic sheet.

6.73 DeleteAxis

DeleteAxis <axis-id>

Deletes the specified axis. If the grid that owns the axis is empty and it is not the only grid in the graph sheet, the grid itself will also be deleted.

The axis will not be deleted if there are curves attached to it.

Parameters

axis-id Axis id as returned by functions [GetAllYAxes\(\)](#), [GetSelectedYAxis\(\)](#) or [GetCurveAxis\(\)](#).

Notes

An axis may only be deleted if it is empty i.e. has no attached curves. Also the main axis may not be deleted.

6.74 DeleteGlobalStyle

DeleteGlobalStyle <name>

Deletes a global style. Note that only user-defined global styles may be deleted. Built-in styles may not be deleted.

6.75 DeleteGraphAnno

DeleteGraphAnno <object-id>

Deletes a graph annotation object such as a curve marker or legend box. See [Graph Objects](#) for details on graph annotation objects.

The command will fail if there are no graphs open. An attempt to delete a graph object which is not an annotation object (e.g. an Axis or Curve) will not raise an error condition, but no action will be taken.

[DeleteGraphObject](#) is a general purpose command that may be used to delete any graph object.

Parameters

object-id Id of object to be deleted.

6.76 DeleteGraphObject

DeleteGraphObject <object-id>

Deletes a graph object such as an axis, curve or annotation object such as a curve marker or legend box. See [Graph Objects](#) for details on graph objects.

If the object being deleted is a graph, all contained objects will also be deleted. The graph is not required to be empty.

If the object being deleted is a grid, all contained axes and curves will be deleted.

If the object being deleted is an axis, the delete operation will fail unless the axis has no attached curves and is not the last axis of its type (X or Y) in its associated grid.

The [DeleteAxis](#) command may be used to delete an axis along with its associated grid.

The command will fail if there are no graphs open.

Parameters

<i>object-id</i>	Id of object to be deleted. This can be obtained from a number of functions. See Graph Objects for a complete list.
------------------	---

See Also

[DeleteGraphAnno](#)

[DeleteAxis](#)

[DelCrv](#)

6.77 DeleteShortWires

DeleteShortWires

Deletes wires that are placed directly across two terminal instances.

6.78 DeleteSymbolProperty

DeleteSymbolProperty <property-name>

Deletes the specified property from a symbol editor symbol.

Parameters

<i>property-name</i>	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.
----------------------	---

6.79 DeleteToolBar

DeleteToolBar <context-name><toolbar-name>

Deletes toolbar definition with given name and context.

Parameters

<i>context-name</i>	Name of window where toolbar is to reside. Refer to CreateToolBar for details
<i>toolbar-name</i>	User assigned name for toolbar. Refer to CreateToolBar for details

Notes

Changes to the toolbar configuration of a window type (i.e. context), do not take effect until all windows of that type have been closed.

See Also

[CreateToolBar](#)

6.80 DelGroup

DelGroup [/cleanUp] [/noDelete] /all | <Group-Name>[Group-Name] ...

Deletes specified groups. See [Groups](#) for more information.

Parameters

<i>/all</i>	If specified all groups except the user group are destroyed.
<i>/cleanUp</i>	Inhibits delete of associated temporary data file. This file will only be deleted any way if the option variable DataGroupDelete is set to OnDelete.
<i>/noDelete</i>	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.

See Also

[CreateGroup](#)

[OpenGroup](#)

[Groups\(\)](#)

6.81 DelLegendProp

DelLegendProp <curve-id>(legend-name)

Delete a graph measurement object defined by its curve and label property.

In version 8.1 and earlier, measurement values were displayed below the curve's legend and were referred to as "Legend Properties". From version 8.2, these have been replaced by Measurement objects. Measurement objects are more advanced and have greater functionality but may perform the same function as the older legend properties.

DelLegendProp can be used to delete a measurement object defined by its associated curve and label property.

Measurement objects may be deleted directly using DeleteGraphObject

Parameters

<i>curve-id</i>	Id of curve, histogram or scatter plot which possesses property. Curve id is returned by the functions GetSelectedCurves() , GetAxisCurves() and GetAllCurves() .
<i>property-name</i>	Name of property to be deleted. The function GetLegendProperties() returns legend properties owned by a specified curve.

6.82 DelMenu

DelMenu [/bypos <pos>] [/force] [/keepid] <menuname>

Deletes specified menu item, or submenu.

Parameters

<i>/bypos</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>menuname</i>	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 for details on menu names.	

6.83 DelProp

DelProp <property name>

Delete specified property from selected schematic instances.

The optional arguments can be used to filter which schematic elements are to have the requested property deleted from them.

Parameters

<i>/handle</i>	If set, filters elements by those that contain the specified handle value.
<i>/prop</i>	If set, filters elements that contain a property with the given name.
<i>/propval</i>	If set, filters elements that contain a property with the given name (arg1) and given value (arg2).
<i>property name</i>	Name of property to be deleted.

6.84 DelSym

DelSym [/local] <symbol-name>

Deletes a schematic symbol from the global library or from the current schematic.

Parameters

<i>/local</i>	Whether to remove from the local symbol library or not.
---------------	---

See Also

[Schematic Symbol Script Definition](#)

6.85 DestroyRunningDialog

DestroyRunningDialog

Destroys a process running dialog

See Also

[CreateRunningDialog](#)

[UpdateRunningDialog](#)

6.86 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'.

6.87 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.

Parameters

<i>/vec</i>	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.

6.88 Display

Display [/file <filename>] [/append <filename>] [/notype] [/notitle] [/type <type>]

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.

Parameters

<i>/append</i>	Append result to <i>filename</i>
<i>/file</i>	Output result to <i>filename</i>
<i>/list</i>	
<i>/noTitle</i>	Do not display the header showing the group name
<i>/notype</i>	Do not list the data type
<i>/type</i>	Filter result according to type. type is a list of typenames separated by ‘ ’. Possible values are: real complex string alias

See Also

[Expressions](#)

6.89 DrawArc

DrawArc [*<theta>*][*<v-over-h>*]

Initiates “arc draw” mode in the currently highlighted symbol editor. This is an interactive command.

Parameters

<i>theta</i>	Swept angle in degrees (integer). Default = 90
<i>v-over-h</i>	Vertical radius/Horizontal radius. Default = 1 (circle)

6.90 DrawArrow

DrawArrow [/loc <x1><y1><x2><y2>]

Draws an arrow in the schematic editor, as specified by the *loc* parameter..

Parameters

/loc Defines the positioning of the arrow as 4 integer values relating to position within the schematic: x1, y1, x2, y2.

6.91 DrawPin

DrawPin [*/forcerepeat*] [*/loc* <x><y>] [(*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.

Parameters

/forceRepeat If specified, the operation will be repeated until the user cancels with the right mouse button. Each new pin be named according to the base name appended with an integer to make it unique.

/loc

base-name 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.

6.92 Echo

Echo <text>

Echoes text to the message window or to a file

Parameters

/append If present *text* is appended to *filename*. If *filename* does not exist, it is created. If the file points to a file that has already been opened for write using [OpenFile\(\)](#), the handle already opened will be used

/box Text is output inside a box composed of asterix characters. This is useful for titles and headings. Currently only works correctly when used with */file* or */append*.

/debug

/file If present *text* is output to *filename*. If *filename* exists, it is overwritten.

/handle File handle as returned by the function [OpenFile\(\)](#). Text will output the file referenced by this handle.

/html If present *text* is assumed to be html formatted.

/list

/page

Prefixes output with a ASCII form feed character.

6.93 EditColour

EditColour <colour-name><colour-spec>

Changes the spec for the named colour object. Named colour objects are simply option variables used to store colour information. This command will write the value to the variable in the form #rrggbb where rr, gg and bb are two digit hex values representing the magnitude of the red, green and blue components respectively.

Parameters

colour-name

Name of colour object. May be the name of a pre-v8 graph colour object. See notes.

colour-spec

Text string that defines the colour. The functions [GetColourSpec\(\)](#) and [SelectColourDialog\(\)](#) return colour spec values. A value in the form #rrggbb or the name of another colour object may also be entered.

Notes

Note that version 7.2 and earlier stored colour information differently and used different names. This command will still recognise the names of colour objects used for graphs and set the correct new colour object. Schematic colour objects used in version 7.2 and earlier are not supported. Refer to documentation on schematic styles in *User's Manual/Schematic Editor/Styles*.

6.94 EditCopy

EditCopy

Copies selected schematic items to clipboard for pasting to SIMetrix and other applications.

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 command [CopyClipSchem](#) in that only selected items are copied. Further, schematics copied with CopyClipSchem can only be pasted into other applications.

Parameters

/mono

If specified, the image obtained when pasting to other applications will be monochromatic. This switch has no effect when pasting to SIMetrix windows.

See Also

[EditPaste](#)
[CopyClipSchem](#)

6.95 EditCut

EditCut

Deletes selected components and places them in the clipboard. Equivalent to the sequence:

[Detach](#)
[EditCopy](#)
[Delete](#)

Parameters

/mono

6.96 EditFile

EditFile

This is an alias to the command [OpenBasicTextEditor](#).

6.97 EditFont

EditFont <font-name><font-spec>

Changes the spec for the named font object.

Parameters

<i>font-name</i>	Name of font object. This can be any of the names returned by the function GetFonts() . (These are listed when the menu File Options Font... is selected.)
<i>font-spec</i>	Text string that defines the font. The functions GetFontSpec() and SelectFontDialog() return font spec values.

6.98 EditGroupTitle

EditGroupTitle <group-name><group-title>

Edit a group's title

6.99 EditPaste

EditPaste

Pastes items from clipboard to a schematic sheet. Only items copied by SIMetrix (using the command [EditCopy](#)) may be pasted, with the exception of text and pictures into the schematic and symbol editors.

6.100 EditPin

EditPin [/name <new-pin-name>] <symbol-name><pin-number>

Edit a pin name of a symbol in the currently installed symbol library.

Parameters

<i>/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.

6.101 EndAllInteractions

EndAllInteractions

Cancels all interactions occurring in all editors

6.102 EndSym

EndSym

EndSym is a Symbol Definition Command. All symbol definitions must end with this command and begin with the command [CreateSym](#).

See Also

[Schematic Symbol Scription Definition](#)

6.103 Execute

Execute [/echo] <command>

Run the script or command *command*.

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, for example if its selected from a file dialog box.
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.

Parameters

<i>/allowextbi</i>	
<i>/echo</i>	Command is copied to the command history drop down box in the command shell.
<i>/literal</i>	Indicates the text in <i>command</i> should be read literally. This switch should be used if the complete command along with any arguments are stored in a variable, to be accessed by Execute through braced substitution. See the example for further explanation.
<i>/startup</i>	Used by initialisation scripts to indicate that a command is being executed on startup. The function CommandStatus() can be used to test this state. This switch must not be used in user scripts.
<i>command</i>	Command to be executed with arguments if required. See <i>/literal</i> above for more information.

Example

Use of the literal flag. If you have a script where a command to execute is contained within a variable, for example:

```
Let command = `inst npn`
```

Then the literal flag should be used to enable the following braced substitution to work:

```
Execute /literal {command}
```

Here is another example of using the literal flag. 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.

6.104 ExecuteMenu

ExecuteMenu <menu-identifier>

Executes the menu with the given full identifier. These identifiers should match those used to create the menu in [DefMenu](#).

See Also

[DefMenu](#)

6.105 FileViewCleanUpFileWatchers

FileViewCleanUpFileWatchers

Removes unnecessary file watchers.

File watchers are created by the *File View* to keep track of when changes to a directory occur. These watchers ensure the *File View* is kept up-to-date, however in some circumstances it may be beneficial to release the system resources used by file watchers that are not deemed necessary.

Generally this operation occurs automatically.

6.106 FloodFillSymbol

FloodFillSymbol

Flood fills a symbol, either interactively or from a specific point. Default behaviour is interactive mode.

Parameters

/loc Defines the location to attempt a flood fill.

6.107 Focus

Focus [/named <window-name>] [/userid <window-id>] [schem|graph]

Focus on a window.

Only one of the options can be used at a time.

Parameters

schem|graph Currently or most recently selected schematic or graph window receives input focus.

6.108 FocusCommandShell

FocusCommandShell

Selects the Command Shell and assigns it keyboard focus.

6.109 FocusShell

FocusShell

Selects the Command Shell and assigns it keyboard focus.

6.110 GroupSelected

GroupSelected

Groups all selected schematic elements. If the selected elements include a set of elements within a group, a hierarchy of groups are created.

6.111 Help

Help [/file <filename>] /contents | /context <context-id>| <topic>

Opens the SIMetrix help system.

Parameters

<i>/contents</i>	Opens help in main contents page
<i>/context</i>	Included only for backward compatibility. 'Help /context id' does the same as 'Help id'
<i>/file</i>	If specified, help will be obtained from <i>filename</i> . Otherwise help file will be SIMetrix.chm
<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
```

6.112 HideCurve

HideCurve <curve-id>

Hides specified curve.

Parameters

curve-id Id of curve to hide. Curve id is returned by the functions [GetSelectedCurves\(\)](#), [GetAxisCurves\(\)](#) and [GetAllCurves\(\)](#).

See Also

[ShowCurve](#)

6.113 HighlightCurve

HighlightCurve [/clear | /unique] curveId

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.

Parameters

<i>/clear</i>	The specified curve will be unhighlighted.
<i>/unique</i>	The specified curve will be highlighted and all others will be unhighlighted.
<i>curveId</i>	Id of curve to be highlighted (or unhighlighted if <i>/clear</i> is specified)

6.114 HighlightWidget

HighlightWidget <widget-id>

Highlights a particular content view.

Parameters

<i>widget-id</i>	The ID of the content view to highlight.
------------------	--

6.115 Hint

Hint [/help help-context] [/id id] [/icon info|warn|error|question] 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.

Parameters

<i>/help</i>	If specified, the box will show a help button which will display the help topic specified by <i>help-context</i> . This is used in some internal scripts but has limited user application.
<i>/icon</i>	Controls the icon displayed in the hint box. This may be one of: <ul style="list-style-type: none"> 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.
<i>/id</i>	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.
<i>message</i>	Message to be displayed.

6.116 HourGlass

HourGlass

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.

Parameters

<i>/clear</i>	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 <i>/clear</i> switch specified.
<i>/off</i>	
<i>/on</i>	

6.117 ImportSymbol

`ImportSymbol [/loc <x><y>] [/local] [/path <pathname>] [/comp] <name>`

Imports an existing symbol to the currently open symbol editor sheet.

Parameters

<i>/comp</i>	Opens the symbol for a component whose path is specified by <i>name</i> .
<i>/fromschematic</i>	Will load the symbol from the last selected schematic. Used internally.
<i>/loc</i>	If <i>/loc</i> switch specified, the symbol is placed at the location specified by <i>x</i> and <i>y</i> . 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.
<i>/local</i>	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.
<i>/path</i>	If specified, the symbol will be converted to a component to be saved in the file specified by <i>pathname</i> .
<i>name</i>	Symbol name.

Notes

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.

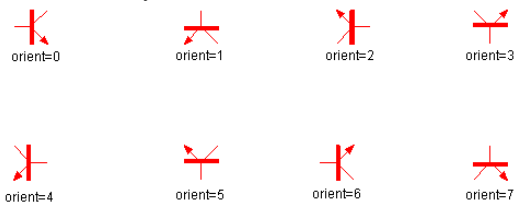
6.118 *Inst*

Inst [*/centre*] [*/select*] [*/repeat*] [*/norepeat*] [*/repeatalways*] [*/loc* *x* *y* *orient*] [*/orient*] [*/comp*] [*/nolocal*] [*/useph*] *symbolname* [*proname*] [*propvalue*]

Places an instance of *symbolname* onto the current schematic. User must press left mouse key to fix the symbol to the schematic.

Parameters

<i>/centre</i>	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.
<i>/comp</i>	Places a component symbol whose path is specified by <i>symbolname</i> .
<i>/loc</i>	If specified, instance is placed directly on sheet without user interaction at the location specified by <i>x</i> and <i>y</i> and orientation specified by <i>orient</i> . 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() .
<i>/nolocal</i>	Only effective if <i>/comp</i> also specified. Forces reloading of the component symbol from the original file instead of using a local cached copy. This may be different if the source file has changed.
<i>/norepeat</i>	If specified a single instance will be placed regardless of the value of the 'RepeatPlace' global option.

<i>/orient</i>	Specifies orientation of symbol. Value from 0 - 7 as illustrated below.
	
<i>/repeat</i>	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 Repeat-Place 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 <i>/norepeat</i> isn't present. If the <i>/loc</i> switch is present repeat action is disabled in all circumstances.
<i>/repeatalways</i>	If specified, instance is placed directly on sheet without user interaction at the location specified by <i>x</i> and <i>y</i> and orientation specified by <i>orient</i> . 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() .
<i>/select</i>	If specified, the instance is selected after being placed on the schematic.
<i>/useph</i>	Only effective if <i>/comp</i> also specified. Will place a place holder symbol if the component symbol specified is not found. Without this switch an error message will be displayed if a component symbol is missing.
<i>symbolname</i>	Name of symbol. Symbol names "caption" and "free_text" are treated specially. See notes.
<i>propname</i>	If specified, property of this name is changed to <i>propvalue</i> .
<i>propvalue</i>	See above.

Notes

The symbol name 'caption' will instantiate the built-in caption annotation object and not a symbol called 'caption'. Similarly the symbol name 'free_text' will instantiate the built-in free text annotation object. In both cases the text displayed will be the property value given. The property name will be ignored. If no property value is provided, the default values "Caption" and "Text" will be used respectively.

6.119 KeepGroup

KeepGroup [*<group-name>*][*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.

6.120 Let

Let [*<vector-expression>*]

Evaluates a vector expression.

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.

Parameters

vector-expression Vector expression to be evaluated. Information on vector expressions can be found in [Expressions](#).

6.121 Listing

Listing [*/error*] [*/filename <filename>*] [*/append <filename>*] [*/anno*]

Displays or outputs to a file a listing of the current netlist.

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.

Parameters

/anno

/append

Result is appended to file of name *filename*.

/converted

/errors

Only lines that are in error are output.

/fileName

Result is written to file of name *filename*.

/icap

/pspice

/spice

6.122 ListModels

ListModels *<filename>*

Generates a dictionary of all models and subcircuits currently available to the simulator (e.g. installed with menu **File | Model Libraries | Add/Remove Libraries** see *User's Manual/Device Library and Parts Management*). 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.

Parameters

/name

6.123 ListOptions

ListOptions <filename>

List all global options to file. Global options are set using the command [Set](#).

Listing contains one line per option with each line being a semi-colon delimited list in the following form:

```
name;type;default-value
```

where:

<i>name</i>	Name of option
<i>type</i>	Type of option. One of 'bool', 'string' or 'real'
<i>default_value</i>	Default value if not set, or if unset using command UnSet .

Parameters

filename File to receive options

6.124 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.

6.125 LoadSimulatorStyleSheet

LoadSimulatorStyleSheet

Applies a style sheet to simulator GUI elements.

Parameters

<i>/clear</i>	Clears the current style sheet first.
<i>/direct</i>	Loads style sheet directly from input string.
<i>/file</i>	Loads style sheet from a file.

6.126 LoadStyleSheet

LoadStyleSheet

Applies a style sheet to the whole application.

Parameters

<i>/clear</i>	Clears the current style sheet first.
<i>/direct</i>	Loads style sheet directly from input string.
<i>/file</i>	Loads style sheet from a file.

6.127 LocalDeclAll

LocalDeclAll

Declare all variables in a script to be local.

Variables can be defined as global by assigning the variable in the form: `global:name`. Subsequently the variable may be accessed with the name alone without the `global:` prefix. The `LocalDeclAll` command disables this behaviour and thus requires all global variables to be accessed using their fully qualified name. The name used alone will access a local variable of that name.

For example, in the following fragment, the variable `var1` will be displayed as having the value 20 as the global variable is reassigned in the second `Let` command

```
Let global:var1 = 10
Let var1=20
Show global:var1
```

In the following, `var1` will be displayed as having the value 10 which is the original assignment. The second `Let` command in this case will assign a new local variable `var1` distinct from the global variable

```
LocalDeclAll
Let global:var1 = 10
Let var1=20
Show global:var1
```

`LocalDeclAll` remains in force for the current script. It is not inherited by scripts that it calls nor does it continue in force in the calling script after `exit`.

6.128 MakeAlias

MakeAlias <vector-name>

Converts a string variable to an alias.

An alias is a string representing a numeric expression. For more information see [Aliases](#).

Parameters

vector-name Variable to be converted

6.129 MakeCatalog

MakeCatalog <out-catalog><all-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.

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 *all-catalog* and *user-catalog* to obtain information about known parts.

Parameters

/force

/listDups

out-catalog File name for catalog. This must be OUT.CAT for use with browser.

all-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.

6.130 MakeSymbolScript

MakeSymbolScript [/all] [/append] [/sortprops] [/catalog <catalog-name>] <filename>[<symbol-name>...]

Creates a script definition of a symbol or group of symbols. For details of script definitions see [Schematic Symbol Script Definition](#).

Parameters

<i>/all</i>	If specified, scripts for all symbols in the global library will be created.
<i>/append</i>	Result will be appended to specified file.
<i>/catalog</i>	If specified, scripts for all symbols in the specified catalog of the global library will be created. This overrides <i>/all</i> .
<i>/sortProps</i>	If specified, all visible properties are ordered alphabetically in the output script. Properties are defined with the command AddProp .
<i>filename</i>	Path of file to be written.
<i>symbol-name</i>	Name of symbol to be scripted. Any number may be specified. If <i>/all</i> or <i>/catalog</i> are specified, this argument will be ignored. If they are not this argument becomes compulsory.

6.131 MakeTree

MakeTree <pathname>

Creates the specified directory path. Unlike the MD command, MakeTree will create any subdirectories required to make the whole path.

6.132 MCD

MCD <directory-name>

Makes a directory and sets it as current (same as MD followed by Cd).

Parameters

directory-name Name of directory to be created.

6.133 MD

MD <directory-name>

Creates a new directory. MD is similar to the DOS MD and MKDIR commands.

Parameters

directory-name Name of directory to be created.

6.134 Message

Message [<message>]

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.

Parameters

message Text to be displayed. If omitted, status window returns to normal view.

6.135 MessageBox

MessageBox <message>[<caption>]

Displays pop-up message box with the specified *message* and *caption*. Note that there is also the function [MessageBox\(\)](#) which is more flexible.

Parameters

message The message to display in the message box.

caption The title caption to use in the message box.

6.136 Move

Move

Initiates the schematic move operation. User interactive command.

Parameters

/mode Specifies editing mode to use for move operation. Options are:

- “default” Use option setting “SchematicMoveMode”
- “ClassicMove” Basic rubberbanding mode
- “Orthogonal” Wires edited so that they remain at right angles as much as possible

6.137 MoveCurve

MoveCurve <curve-id><axis-id>

Moves a curve, histogram or scatter plot to a new y-axis.

Parameters

<i>curve-id</i>	Id of curve, histogram or scatter plot as returned by functions GetSelectedCurves() , GetAxisCurves() and GetAllCurves() .
<i>axis-id</i>	Axis id as returned by functions GetAllYAxes() , GetSelectedYAxis() or GetCurveAxis() .

6.138 MoveFile

MoveFile [/force] <path-1><path-2>

Moves a file from *path-1* to *path-2*.

Parameters

<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.
---------------	--

6.139 MoveMenu

MoveMenu [/bypos position] <menu-path><shift-by>

Moves the position of a menu item by a specified count.

Parameters

<i>/bypos</i>	Value <i>position</i> is an optional number that identifies a menu item by its position within a sub-menu. If this is specified the menupath must identify a sub-menu rather than a menu item.
<i>menu-path</i>	Path to the menu to move, see DefMenu for full details of path names.
<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.

6.140 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.

Notes

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.

6.141 Netlist

```
Netlist [/num] [/subckt] [/nopinames] [/noOutput] [/template] [/sep] [/diag] [/top] [/plain] [/lang]
        [/wireTemplate] [/dotEnd] [/noDescend] [/f11Top] [/simplis] [/nodemap] [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.

Parameters

<i>/diag</i>	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 diag is set to partial, only dangling wires and pins are reported.
<i>/dotEnd</i>	Forces .END to be placed at the end of the netlist.
<i>/f11Top</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>/inhibitTemplateScripts</i>	Inhibits execution of template scripts
<i>/lang</i>	Name of language to be output at the top of the netlist output. This is in the form “*#language” and is used by SIMetrix for compatibility with other simulators. Default is “SIMETRIX”.
<i>/nodemap</i>	Generates SIMPLIS .NODE_MAP controls for user named nets.
<i>/noDescend</i>	Netlister does not descend into hierarchy and processes items at the top level only.
<i>/noOutput</i>	If specified, no netlist output is generated. The net names attached to wires are updated.
<i>/nopinames</i>	If specified, the <i>pinames</i> specifier is not output for X devices. The pin-names 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.
<i>/num</i>	If specified, a SPICE 2 compatible netlist using node numbers is created.
<i>/paramsSeparator</i>	Inhibits execution of template scripts

<i>/path</i>	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.
<i>/plain</i>	Equivalent to <code>/noPinnames /top /lang none</code> .
<i>/selSubOut</i>	Inhibits execution of template scripts
<i>/sep</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>/simplis</i>	Specify this if creating a netlist for use with SIMPLIS. Forces switches: <code>/dotEnd /f11Top /nodemap /num /nopinnames /sort</code> . If <code>/template</code> is not specified, a default of <code>/template simplis_template template</code> will be forced. Finally if <code>/wireTemplate</code> is not specified, a default of <code>/wireTemplate %busname%\$%wirenum%</code> will be forced.
<i>/sort</i>	If specified, the netlist lines will be output in alphanumeric sorted order.
<i>/subckt</i>	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.
<i>/template</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 <i>User’s Manual/Schematic Editor/Template Property</i> .
<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>/wireTemplate</i>	Format for bus wires. <code>wire_template</code> may contain the keywords <code>%BUSNAME%</code> and <code>%WIRENUM%</code> . These resolve to the bus name and wire number respectively. So a spec set to <code>%BUSNAME%#%WIRENUM%</code> would give the default, i.e. bus names like <code>BUS1#2</code> . A spec of <code>%bus-name%[%wirenum%]</code> would give bus names like <code>BUS1[2]</code> .
<i>filename</i>	File to which netlist is written. If not specified, the netlist is displayed in the message window.

6.142 NewAnnotation

NewAnnotation [`/rect`] [`/ellipse` |`/arrow`] |`/line`] [`/roundedrect`] [`/triangle`] [`/octagon`]

Interactive placement of a new annotation. The new annotation type is attached to the cursor, such that when the cursor is next clicked on the schematic the annotation placement begins.

Use one of the flags to set the type of annotation to create.

Parameters

<i>/arrow</i>	Creates an arrow.
<i>/ellipse</i>	Creates an ellipse.
<i>/line</i>	Creates a line.
<i>/octagon</i>	Creates a octangon.
<i>/rect</i>	Creates a rectangle.
<i>/rhombus</i>	Creates a rhombus.
<i>/roundedrect</i>	Creates a rounded rectangle.
<i>/triangle</i>	Creates a triangle.

6.143 NewAxis

NewAxis

Creates a new y-axis. This will be initially empty and selected. See *User's Manual/Graphs, Probes and Data Analysis/Graph Layout* for more information on multiple y-axes.

6.144 NewBasicTextEditor

NewBasicTextEditor

Creates a new plain text document in the SIMetrix environment. Use this for files with no recognised format. Use one of the following commands to create documents with specific formats:

[NewNetlist](#) to create a model file or netlist file

[NewLogicDefinitionEditor](#) to create a logic definition file for the arbitrary logic block

[NewScript](#) to create a script

[NewVerilogA](#) to create a Verilog-A source file

[NewVerilogHDL](#) to create a Verilog-HDL source file

6.145 NewFileView

NewFileView

Creates a new File View and attaches it to the current window. If a File View already exists in the window, this command does nothing.

Parameters

<i>/restore</i>	Identifies this is part of a restore session call, argument is the object name.
-----------------	---

6.146 NewGraphWindow

NewGraphWindow <window-title>

Creates a new graph window to which new graphs may be directed.

6.147 NewGrid

NewGrid

Creates a new grid. See *User's Manual/Graphs, Probes and Data Analysis/Graph Layout* for more information on axes and grids.

Parameters

<i>/detach</i>	Make a detached x-axis. A detached x-axis is not connected to other axes and will zoom independently
<i>/digital</i>	Make a digital grid

6.148 NewLabel

NewLabel <label-text>[/italics] [/bold] [/font <font-family>] [/size <point-size>] [/style <style-name>] [/repeating] [/loc <x><y>]

Adds a new unplaced text label to a schematic. This is an interactive command, with the label being initially attached to the cursor, unless the `loc` flag is set.

If a style is given, that style is applied. If bold, italics, size, or font are given, a new style is created using those. If style is given as well as a font, size, bold or italics option, the given options will override the existing style and a new style will be created for this element.

Parameters

<i>/bold</i>	Uses bold.
<i>/font</i>	Sets the font family, argument is the name of the font family to use.
<i>/italics</i>	Uses italics.
<i>/loc</i>	If set, places the label at the given position.
<i>/repeating</i>	Causes repeated placement of the label until a cancel request is made (right click or escape press).
<i>/size</i>	Sets the font size, argument is point size of the font to use.
<i>/style</i>	Sets the name of the style to apply to the label.
<i>label-text</i>	The text of the label. Use backslash n to set line breaks within the text.

6.149 NewLogicDefinitionEditor

NewLogicDefinitionEditor

Creates a new plain text document in the SIMetrix environment. Use this for files with no recognised format. Use one of the following commands to create documents with specific formats:

- [NewNetlist](#) to create a model file or netlist file
- [NewBasicTextEditor](#) to create a plain text file
- [NewScript](#) to create a script
- [NewVerilogA](#) to create a Verilog-A source file
- [NewVerilogHDL](#) to create a Verilog-HDL source file

6.150 NewNetlist

NewNetlist

Creates a new plain text document in the SIMetrix environment. Use this for files with no recognised format. Use one of the following commands to create documents with specific formats:

- [NewLogicDefinitionEditor](#) to create a logic definition file for the arbitrary logic block
- [NewBasicTextEditor](#) to create a plain text file
- [NewScript](#) to create a script
- [NewVerilogA](#) to create a Verilog-A source file
- [NewVerilogHDL](#) to create a Verilog-HDL source file

6.151 NewPartSelector

NewPartSelector

Creates a new Part Selector and attaches it to the current window. If the window already contains a part selector, the command does nothing.

Parameters

/restore Whether this is a restore session call, argument is the object name to use.

6.152 NewPlot

NewPlot

Plots a curve. Uses existing graph sheet if one is available. Otherwise creates a new graph sheet.

The Plot command is identical to the [Curve](#) command except that it will create a new graph sheet if required. The Curve command never creates a new graph sheet.

For full details refer to [Curve](#) command.

6.153 NewPrinterPage

NewPrinterPage

Advances printer to the a new page. This may be used for customised or noninteractive printing. See [Non-interactive and Customised Printing](#).

6.154 NewSchem

NewSchem [/newWindow] [/simulator simulator] <window-title>

Creates a new schematic sheet within the currently selected schematic window. If no schematic window is open, one will be created.

Parameters

<i>/newWindow</i>	If specified, a new schematic window will be created.
<i>/simulator</i>	Specifies initial simulaor 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).
<i>window-title</i>	The name of the schematic, which will appear in the schematics title bar and will be the default filename that will be used if File Save is used. Note that no file is created by the NewSchem command.

6.155 NewScript

NewScript

Creates a new plain text document in the SIMetrix environment. Use this for files with no recognised format. Use one of the following commands to create documents with specific formats:

[NewNetlist](#) to create a model file or netlist file

[NewBasicTextEditor](#) to create a plain text file

[NewLogicDefinitionEditor](#) to create a logic definition file for the abritrary logic block

[NewVerilogA](#) to create a Verilog-A source file

[NewVerilogHDL](#) to create a Verilog-HDL source file

6.156 NewStyle

NewStyle [/name <style-name>] [/linecolour <hex-bgr-colour>] [/linestyle <pen-style>] [/linethickness <thickness>] [/fontcolour <hex-bgr-colour>] [/fontfamily <family>] [/fontsize <point-size>] [/fontitalics] [/fontbold]

Creates a new style.

Parameters

<i>/fontbold</i>	Switches on the use of bold font.
<i>/fontcolour</i>	The colour of the font, argument in the form <code>0x00bbggrr</code> .
<i>/fontfamily</i>	The font family to use, argument is the name of a font family.
<i>/fontitalics</i>	Switches on the use of italic font.
<i>/fontoverline</i>	Overline the text.
<i>/fontsize</i>	The size of the font, argument is a integer point size.
<i>/fontunderline</i>	Underline the text.
<i>/linecolour</i>	The line colour to use, argument in the form <code>0x00bbggrr</code> .
<i>/linefixedthickness</i>	The colour of the font, argument in the form <code>0x00bbggrr</code> .
<i>/linestyle</i>	The style of the line to use. Options are: <i>Solid</i> , <i>Dash</i> , <i>Dot</i> , <i>Dashdot</i> , <i>Dash-dotdot</i> .
<i>/linethickness</i>	The thickness of the line.
<i>/name</i>	The name of the style, argument is the name.

Notes

Colours are defined as a hex value with blue-green-red specified components in the form `0x00bbggrr`.

6.157 NewSymbol

NewSymbol

Opens a new symbol editor view.

6.158 NewVerilogA

NewVerilogA

Creates a new Verilog A editor.

See Also

[NewNetlist](#) to create a model file or netlist file

[NewBasicTextEditor](#) to create a plain text file

[NewScript](#) to create a script

[NewLogicDefinitionEditor](#) to create a logic definition file for the arbitrary logic block

[NewVerilogHDL](#) to create a Verilog-HDL source file

6.159 NewVerilogHDL

NewVerilogHDL

Creates a new Verilog HDL editor.

See Also

[NewNetlist](#) to create a model file or netlist file

[NewBasicTextEditor](#) to create a plain text file

[NewScript](#) to create a script

[NewVerilogA](#) to create a Verilog-A source file

[NewLogicDefinitionEditor](#) to create a logic definition file for the arbitrary logic block

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

6.160 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.

Use the `/stack` switch to restore painting when the current script completes. Without this the NoPaint state will not be automatically cleared until complete control returns to the command line. This means that if a graph operation using NoPaint is executed while running a simulation, the paint operation may not actually be seen until the simulation is complete or paused.

Parameters

<code>/reenable</code>	Flag to indicate whether to re-enable painting or not. Default is false.
<code>/stack</code>	Instructs to restore NoPaint state to the starting state (either on or off) when current script completes.

6.161 NoUndo

NoUndo

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.

Parameters

<i>/nocapture</i>	Normally NoUndo, saves the current state so that the next undo operation will restore the state to immediately before NoUndo was called. The /no-capture switch inhibits this.
<i>/release</i>	Restores undo buffer save operations. This happens automatically when control returns to the command line.

6.162 OpenAsciiFile

OpenAsciiFile <filename>

Open a schematic ASCII format file for manual text editing. This can be useful for debugging or for some operations that are difficult to perform using the GUI editor.

Parameters

<i>/encoding</i>	encoding. For details see documentation of second argument to LoadFile()
<i>/fws</i>	File watcher status, enable/disable/auto

See Also

[OpenSchem](#)

6.163 OpenBasicTextEditor

OpenBasicTextEditor <filename>

Open a plain text file for manual text editing. This command opens the text file with no syntax highlighting. Use one of the following commands to open files with specific formats:

[OpenNetlist](#) to open a model file or netlist file

[OpenLogicDefinitionEditor](#) to open a logic definition file for the arbitrary logic block

[OpenScript](#) to open a script

[OpenVerilogA](#) to open a Verilog-A source file

[OpenVerilogHDL](#) to open a Verilog-HDL source file

[OpenAsciiFile](#) to open a schematic file in the text editor

Parameters

<i>/encoding</i>	encoding. For details see documentation of second argument to LoadFile()
<i>/fws</i>	File watcher status, enable/disable/auto
<i>filename</i>	Path of text file to open

6.164 OpenDirectory

OpenDirectory <path>

Opens the directory as given by the argument.

Parameters

<i>path</i>	The path of the directory to be opened.
-------------	---

6.165 OpenExternalFile

OpenExternalFile <filename>

Opens the given file path in the operating systems default program associated with that file.

Parameters

<i>filename</i>	The path of the file to be opened.
-----------------	------------------------------------

6.166 OpenGraph

OpenGraph <file name>

Opens the graph file *filename* and displays it. Accepts both the older binary format along with the new XML-based format. If the specified file is a binary graph file, it will first be converted to a temporary XML file.

Parameters

<i>/v32bitfix</i>	
-------------------	--

6.167 OpenGroup

OpenGroup [/text] [/spice2] [/spice3] [/purge] [/overwrite] [/forcereadopen] [/deleteonclose] [/ign] [*<filename>*]

Reads in a data file and 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.

Parameters

<code>/append</code>	
<code>/deleteonclose</code>	If specified, the file will be marked as volatile and will be deleted once it is no longer needed.
<code>/forcereadopen</code>	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.
<code>/ign</code>	
<code>/overwrite</code>	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.
<code>/purge</code>	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 menu Simulator Manage Data Groups... menu (which uses <code>DelGroup</code>).
<code>/simplis</code>	
<code>/spice2</code>	If specified, <code>filename</code> will be read in as a SPICE2 raw file as generated by SPICE2g.6. This is an unsupported feature.
<code>/spice3</code>	If specified, <code>filename</code> , will be read in as a SPICE3 raw file. <code>OpenGroup</code> will read in the whole file into RAM. This may be inappropriate if the file is large. The command OpenRawFile is usually a better choice for reading SPICE3 raw files as this rewrites the data to a native data file for access on demand.
<code>/text</code>	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 <code>SaveGroup</code> command. See Data Files Text Format for full details on the text format.
<code>filename</code>	Name of file to be input. If not specified, an open file dialog box will be opened allowing the user to choose from available files.

See Also

[CreateGroup](#)
[DelGroup](#)
[SaveGroup](#)
[Groups\(\)](#)

6.168 OpenLinkedRun

OpenLinkedRun

Open a linked simulation run. Call this function prior to the [Run](#) command in order to link them together. Linked runs create multi-division vectors in a single data group and are useful when sweeping some variable.

A linked run may be closed using [CloseLinkedRun](#). It will also automatically close when control returns to the command line

Parameters

<i>/labelprefix</i>	Creates default label for each linked run. This is usually defined using <i>/label</i> with the Run command, but if omitted will instead default to <i>label-pref=runnumber</i> where <i>runnumber</i> is a number starting at 1 and incrementing by 1 for each new run. Default is "Run"
---------------------	---

See Also

[CloseLinkedRun](#)

6.169 OpenLogicDefinitionEditor

OpenLogicDefinitionEditor <filename>

Opens a logic definition file for the arbitrary logic device in the text editor. This will apply syntax highlighting for the logic definition language.

Parameters

<i>/encoding</i>	encoding. For details see documentation of second argument to LoadFile()
<i>/fws</i>	File watcher status, enable/disable/auto
<i>filename</i>	Path of logic definition file to open

See Also

- [OpenNetlist](#) to open a model file or netlist file
- [OpenBasicTextEditor](#) to open a plain text file
- [OpenScript](#) to open a script
- [OpenVerilogA](#) to open a Verilog-A source file
- [OpenVerilogHDL](#) to open a Verilog-HDL source file
- [OpenAsciiFile](#) to open a schematic file in the text editor

6.170 OpenNetlist

OpenNetlist <filename>

Opens a SPICE netlist or model file in the text editor. This will apply syntax highlighting for the simulator command language.

If the option setting `NetlistEditor` is set and points to a valid executable file, this command will instead call the specified application to open the file.

Parameters

<i>/encoding</i>	encoding. For details see documentation of second argument to LoadFile()
<i>/fws</i>	File watcher status, enable/disable/auto
<i>filename</i>	Path of netlist or model file to open

See Also

[OpenLogicDefinitionEditor](#) to open a logic definition file for the arbitrary logic block

[OpenBasicTextEditor](#) to open a plain text file

[OpenScript](#) to open a script

[OpenVerilogA](#) to open a Verilog-A source file

[OpenVerilogHDL](#) to open a Verilog-HDL source file

[OpenAsciiFile](#) to open a schematic file in the text editor

6.171 OpenPrinter

OpenPrinter [*/portrait*] [*/numCopies* <num-copies>] [*/index* <index>] [*/title* <title>] [*/printer* <printer>]
 [*/greyscale* on/off]

Starts a print session. This may be used for customised or non-interactive printing. See [Non-interactive and Customised Printing](#)

Parameters

<i>/greyscale</i>	Set to 'on' to enable grey-scale printing
<i>/index</i>	Printer to use. This can be found from the function GetPrinterInfo() . If omitted, the application default printer will be used.
<i>/numCopies</i>	Number of copies to print.
<i>/portrait</i>	If specified, print will be in portrait orientation, otherwise it will be landscape
<i>/printer</i>	Specify printer by name. If omitted, printer will be defined by its index (see below) or the application default printer will be used.

/title Title of *print job*. 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. *title* is not printed on the final document.

6.172 OpenRawFile

OpenRawFile [*/purge*] [*/bufsize* *buffer_size*] [*/spice2*] *rawfile* [*datafile*]

Opens a SPICE 3 format ASCII raw file.

Parameters

/bufsize Specifies the percentage proportion of installed RAM that is used for buffering the data. See Notes below for more details. Default value is 10 (%).

/csdf If specified, the *datafile* is assumed to be in CSDF format.

/purge 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 **Simulator | Manage Data Groups...** menu.

/spice2 If specified, *datafile* is assumed to be in SPICE format. This is an unsupported option.

rawfile Raw file to open.

datafile 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 TempDataDir 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 *Simulator Reference Manual/Running the Simulator/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.

6.173 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.

Parameters

<i>/backup</i>	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 as a backup. This switch is used for the save/restore session feature and for recovering auto-saved schematics after an unexpected program exit.
<i>/cd</i>	If specified, the directory holding <i>filename</i> is made current.
<i>/readonly</i>	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.
<i>filename</i>	The name of the file to load the schematic from.

6.174 OpenScript

OpenScript <filename>

Opens a script source file in the text editor. This will apply syntax highlighting for the script language along with prompts for function names and commands.

If the option setting `ScriptEditor` is set and points to a valid executable file, this command will instead call the specified application to open the file.

Parameters

<i>/encoding</i>	encoding. For details see documentation of second argument to LoadFile()
<i>/fws</i>	File watcher status, enable/disable/auto
<i>filename</i>	Path of script source file to open

See Also

[OpenNetlist](#) to open a model file or netlist file

[OpenBasicTextEditor](#) to open a plain text file

[OpenLogicDefinitionEditor](#) to open a logic definition file for the arbitrary logic block

[OpenVerilogA](#) to open a Verilog-A source file

[OpenVerilogHDL](#) to open a Verilog-HDL source file

[OpenAsciiFile](#) to open a schematic file in the text editor

6.175 **OpenSimplisStatusBox**

OpenSimplisStatusBox

Opens the SIMPLIS simulation status box.

See Also

[CloseSimplisStatusBox](#)

6.176 **OpenVerilogA**

OpenVerilogA *<filename>*

Opens a Verilog-A source file in the text editor. This will apply syntax highlighting for the Verilog-A language.

Parameters

<i>/encoding</i>	encoding. For details see documentation of second argument to LoadFile()
<i>/fws</i>	File watcher status, enable/disable/auto
<i>filename</i>	Path of Verilog-A source file to open

See Also

[OpenNetlist](#) to open a model file or netlist file

[OpenBasicTextEditor](#) to open a plain text file

[OpenScript](#) to open a script

[OpenLogicDefinitionEditor](#) to open a logic definition file for the arbitrary logic block

[OpenVerilogHDL](#) to open a Verilog-HDL source file

[OpenAsciiFile](#) to open a schematic file in the text editor

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

6.177 **OpenVerilogHDL**

OpenVerilogHDL *<filename>*

Opens a Verilog-HDL source file in the text editor. This will apply syntax highlighting for the Verilog-HDL language.

Parameters

<i>/encoding</i>	encoding. For details see documentation of second argument to LoadFile()
<i>/fws</i>	File watcher status, enable/disable/auto
<i>filename</i>	Path of Verilog-HDL source file to open

See Also

[OpenNetlist](#) to open a model file or netlist file

[OpenBasicTextEditor](#) to open a plain text file

[OpenScript](#) to open a script

[OpenVerilogA](#) to open a Verilog-A source file

[OpenLogicDefinitionEditor](#) to open a logic definition file for the arbitrary logic block

[OpenAsciiFile](#) to open a schematic file in the text editor

Product

SIMetrix and SIMetrix/SIMPLIS Pro and Elite

6.178 OpenWebPage

OpenWebPage <URL>

Opens a web page in the system default browser. Argument must be the full path URL.

6.179 OptimiserCreate

OptimiserCreate [file]

Creates an optimiser widget (GUI) from an XML definition file

6.180 OptimiserRequestFinish

OptimiserRequestFinish

Requests that current optimisation analysis is terminated. Optimiser will complete after end of current simulation

Note that this command can only be executed by an assigned key or menu with the direct execution option specified.

6.181 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.

6.182 Pan

Pan $\langle x \rangle \langle y \rangle$

Pan (scroll) schematic specified number of grid squares.

Parameters

<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.

6.183 PasteGraphImageToSchematic

PasteGraphImageToSchematic

Copies a picture of the last selected graph to the last selected schematic. When placed on the schematic, the image can be stretched to the required size.

Parameters

<i>/size</i>	Specifies the resolution of the image to capture. Values are width and height. Values too small may cause parts of the graph to disappear. Default values are 400 300.
--------------	--

6.184 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](#).

A paused simulation can be restarted with the command [Resume](#).

6.185 PlaceCursor

PlaceCursor [/main x-main y-main] [/datum x-datum y-datum]

Positions graph cursors if they are enabled.

Parameters

<i>/datum</i>	Location of reference cursor. Position is determined by <i>x-datum</i> . <i>y-datum</i> is only used for non-monotonic curves (e.g. nyquist plots) where there is more than one y value for a given x value.
<i>/main</i>	Location of main measurement cursor. Position is determined by <i>x-main</i> . <i>y-main</i> is only used for non-monotonic curves (e.g. Nyquist plots) where there is more than one y value for a given x value.

6.186 Plot

Plot

Plot can be used to add a new curve to an existing graph created with Plot or to change the way it is displayed.

The Plot command is identical to the [Curve](#) command except that it always creates a new graph sheet unless the current sheet is empty. The Curve command never creates a new graph sheet.

For full details refer to [Curve](#) command.

Parameters

/newSheet

6.187 PreProcessNetlist

PreProcessNetlist

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.

Some SIMetrix models do make use of the pre-processor. For example the multi-level capacitor and inductor models employ the pre-processor. Placing 'vars:' followed by any parameters at the end of a SIMetrix subcircuit call will result in the subcircuit model being pre-processed.

Documentation for the pre-processor language syntax may be found in *SIMPLIS Reference Manual/Running SIMPLIS/Netlist Preprocessor*.

6.188 PrintGraph

PrintGraph [/caption <caption>] [/margin l t r b] [/major on|off] [/minor on|off] [/mono] [dim-left, dim-top, dim-right, dim-bottom]

Prints the current graph sheet.

Parameters

<i>/caption</i>	Caption printed at the bottom of the page.
<i>/interactive</i>	
<i>/major</i>	Specify whether major grid lines should be printed. options are 'on' or 'off'. Default is 'on'.
<i>/margin</i>	Page margins in mm, stated in the form <i>left, top, right, bottom</i> .
<i>/minor</i>	Specify whether minor grid lines should be printed. options are 'on' or 'off'. Default is 'on'.
<i>/mono</i>	If specified, the graph will be printed in black and white.
<i>/nointeractive</i>	
<i>dim-left, dim-top, dim-right, dim-bottom</i>	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.

6.189 PrintSchematic

```
PrintSchematic [/caption <caption>] [/fixed <grid-size>] [/margin l t r b] [/mono on|off]
                [<dim-left><dim-top><dim-right><dim-bottom>]
```

Prints the current schematic.

Parameters

<i>/caption</i>	Caption printed at the bottom of the page.
<i>/fixed</i>	If specified, fixed scaling will be used. <i>grid-size</i> 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 <i>isotropic</i> . That is the aspect ratio will be maintained.
<i>/margin</i>	Page margins in mm, stated in the form <i>left, top, right, bottom</i> .
<i>/mono</i>	If specified, the graph will be printed in black and white.

dim-left, *dim-top,* Dimensions and position of printed image on page. Values are relative to
dim-right, *dim-* page size less the specified margins in units equal to 1/1000 of the page
bottom 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.

6.190 Probe

Probe [/type 1|2|P|N] [<probe-message>]

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.

Parameters

/type 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

6.191 Prop

Prop Prop [/hide/show/toggle] [/flags <attrib. flags>] [/noAdd] [/showName] [/hideName] [/code 0|1|2|3]
[/overrideStyle styleName] <name>[<value>]

Modifies a property value of a schematic component if it exists. If it doesn't exist the property is added.

Parameters

/all
/code

<i>/flags</i>	Argument is either a value or a property. If a value is specified, it 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. If a property is specified, it 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 flags define the property's attributes.
<i>/handle</i>	
<i>/hide</i>	Make property invisible.
<i>/hideName</i>	If specified, the name of the property will be hidden.
<i>/hideNew</i>	Hide property value if a new property is being added. If the property already exists, its visibility will remain unchanged.
<i>/noAdd</i>	If specified, property will not be added if the instance does not already possess it.
<i>/order</i>	Override of order for auto positioned properties. Set this value to be 0 or above to manually adjust property order. Set to -1 to revert back to default ordering.
<i>/overridestyle</i>	Style name to use, overriding any styles inherited from the parent instance. If unset or the value is "", the style of the instance it is associated with will be used.
<i>/pinloc</i>	If specified, the property will be positioned at a fixed location next to the pin specified by pinnumber.
<i>/prop</i>	
<i>/propval</i>	
<i>/show</i>	Make property visible.
<i>/showName</i>	If specified, the name of the property will be made visible along with its value in the form "name=value".
<i>/toggle</i>	

Notes

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 table:

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, where baseline means the base for upper case characters, the tails of some lower case characters go below the baseline:
	00 top
	01 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.

Example

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
```

6.192 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.

6.193 Quit

Quit

Terminates SIMetrix. If there are any modified schematics open, the user will be prompted to save them first.

6.194 RD

RD <directory-name>

Remove a directory. Rd is similar to the DOS RD and RMDIR commands.

Parameters

directory-name Name of directory to be removed.

6.195 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 CD in directory Docs/Manuals/Supporting Files. 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.

Parameters

filename Logic compatibility file

6.196 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.

6.197 Redirect

Redirect /err/out [*<filename>*]

Redirects messages (i.e. text which is normally displayed in the message window) to a file. One or both of */err* or */out* must be specified.

Parameters

<i>/err</i>	Specifies that error and warning messages are to be redirected.
<i>/out</i>	Specifies that messages other than errors and warnings are to be redirected.
<i>filename</i>	Name of file to which messages are sent. If not specified messages are sent to the message window the specified redirect mode is cancelled and messages of that type will be sent to the command shell.

See Also

[RedirectMessages](#) redirects everything to a file.

6.198 RedirectMessages

RedirectMessages on *<filename>* | dup *<filename>* | off | flush

Redirects all command shell messages to a file. Everything that would normally be displayed in the command shell such as error messages will be sent to the specified file. An option is available to copy command shell output to a file, that is the command shell messages continue to be displayed but are also written to a file.

Note that some messages are sent in HTML format to show bold text and other formatting. These will be shown in the file in their native form including the HTML tags.

Parameters

<i>on</i>	Switch on redirect. All messages will go to <i>filename</i> and no output will appear in the message window
<i>append</i>	As on but appends text to file if it already exists
<i>dup</i>	Switch on redirect. All messages will go to <i>filename</i> and to the message window
<i>off</i>	Switch off redirect. Restore message output to command shell and close redirect file
<i>flush</i>	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

6.199 RedrawGraph

RedrawGraph <graph-id>

Redraws the chosen graph.

Parameters

graph-id The ID of the graph to redraw.

6.200 RegisterUserFunction

RegisterUserFunction <Function-Name><Script-Name>[(*min-number-args*)] [(*max-number-args*)]

Creates a user defined function based on a script.

Parameters

<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](#).

6.201 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.

Parameters

<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>/log</i>	If specified, all renamed models will be listed in <i>logfile</i> .
<i>/report</i>	If specified a report of progress will be displayed in the command shell.
<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-catalogfile</i> , any user association of renamed models will be appropriately modified.
<i>user-catalog-file</i>	If specified a report of progress will be displayed in the command shell.

6.202 RenameMenu

```
RenameMenu <menu-path><new-item-name>
```

Renames a menu item.

Parameters

<i>menu-path</i>	The full path of the menu to change the name for.
<i>new-item-name</i>	The new name to use for the menu item.

6.203 RepeatLastMenu

```
RepeatLastMenu <window-name><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.

Parameters

<i>window-name</i>	Identifies the window type that owns the menu. See DefMenu for list of possible values.
<i>top-menu-name</i>	The top level menu name. This is the name that appears in the menu bar.

6.204 ReplayTraces

ReplayTraces <group-name>

The definitions for graph curves that are created by fixed probes are stored in the simulation data group. Normally these are automatically executed when the simulation is run. This command can be used to execute those curve definitions at any later time.

Parameters

<i>groupname</i>	Name of group from which the fixed probe definition will be retrieved
------------------	---

6.205 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.

6.206 ResizeWindow

ResizeWindow /width [width] /height [height]

Resizes the current window.

Parameters

<i>/height</i>	The height in pixels to use.
<i>/width</i>	The width in pixels to use.

6.207 RestartTran

RestartTran <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.

Parameters

stop-time The restarted run will continue until it reaches this time.

6.208 RestoreCommandShell

RestoreCommandShell

Re-opens the command shell if closed or brings the command shell to the front if it is not visible.

Parameters

/force If set, this will force the command shell to appear in the currently selected window.

6.209 RestoreDefaultStyles

RestoreDefaultStyles [/all] [/selected]

Restores default styles. One of the switches must be applied, otherwise no changes will be made.

Parameters

/all All elements are changed back to their default style settings and any overrides to the default styles are removed.

/selected Causes the selected elements to be changed back to their default styles. Any overrides of these styles are not checked for or removed.

6.210 Resume

Resume

Resumes a previously paused simulation.

6.211 RotInst

RotInst [<orientation>]

Changes orientation of selected items.

Parameters

<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

6.212 Run

Run [/check] [/an <analysis-spec>] [/options <options-string>] [/optforce <options-string>] [/list <list-file>] [/local] [/nolist] [/force] [/label <division-label>] [/append <group-name>] [/pauseAt <pause-time>] [/noData] [/noStatus] [/sweep start|continue|finish] [/cd <directory>] [/extraLine <extra-line>] netlist [datafile]

Runs a simulation on specified netlist.

Parameters

<i>/an</i>	If specified, any analysis controls (e.g .TRAN, .AC etc.) in the netlist are ignored and the statement in <i>analysis-spec</i> is executed instead.
<i>/append</i>	Append data created to <i>group-name</i> which would always be the data group created by the first run in the sequence. ‘/sweep continue’ or ‘/sweep finish’ 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.
<i>/cd</i>	Simulator process current working directory is set to <i>directory</i> . If not specified the current working directory is set to the location of <i>netlist</i>
<i>/check</i>	Performs a check on the netlist for syntax errors but does not run the simulation
<i>/extraLine</i>	Adds <i>extra-line</i> to the end of the netlist. Use .include to append multiple lines
<i>/force</i>	<i>datafile</i> will be overwritten if it already exists. Otherwise an error message will be displayed.
<i>/label</i>	Used with /sweep to name the division of a linked run.
<i>/list</i>	Override default name for list file with <i>list-file</i> . For multi-core multi-step runs, this is applied only to the primary list file

<i>/local</i>	Save data using simulator local process. Normally data is sent through the front end.
<i>/noData</i>	Only data explicitly specified by .PRINT or .KEEP controls will be output. Usually all top level data is saved. Equivalent to placing ".KEEP /nov /noi /nodig" in netlist.
<i>/noHistory</i>	Disables fixed probe history feature. Equivalent to adding separate-Curves=true to the .GRAPH statement
<i>/nolist</i>	Inhibits creation of list file.
<i>/nostart</i>	Similar to /check but proceeds further into the simulation run sequence but stop short of starting the run. Will output message 'User aborted analysis'
<i>/noStatus</i>	Inhibits update of the GUI status box. Use in conjunction with /nofocus to run a hidden simulation
<i>/optcommand</i>	Optimiser command file. If present, all analysis and optimiser definitions are read from this file. Any such lines found in main netlist will be ignored
<i>/optforce</i>	Same as /options but overrides any .OPTIONS setting in the netlist
<i>/pauseAt</i>	Pauses simulation at first time point after <i>pause-time</i> . Applied only to primary process in multi-core multi-step run.
<i>/sweep</i>	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 '/sweep start' while the final run should specify '/sweep finish'. All intermediate runs should specify '/sweep continue'. All runs except the first must also specify '/append'
<i>netlist</i>	Input netlist filename
<i>datafile</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 one of two methods. The traditional method compatible with version 8.1 and earlier is to by using the /sweep, /append and /label switches. From 8.2 a newer and simpler method is to use the [OpenLinkedRun](#) and [CloseLinkedRun](#) commands. 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.

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 using OpenLinkedRun/CloseLinkedRun

```

** Open linked Run
OpenLinkedRun

Run /label "Run=1" netlist.net
... changes to netlist

** second run
Run /label "Run=2" netlist.net
... changes to netlist

** third run
Run /label "Run=3" netlist.net
... changes to netlist

** fourth and final run
Run /label "Run=4" netlist.net

CloseLinkedRun

```

In the above, the /label switches are not actually necessary as "Run=n" labels will be generated automatically. Any label, preferably in the form name=value may be used.

Linked Run Example using /sweep and /append

This is the old method. The newer OpenLinkedRun/CloseLinkedRun is simpler.

```

** First run
Run /sweep start /label "Run=1" netlist.net
** save group name
Let grp1 = (Groups())[0]
... changes to netlist
** second run
Run /sweep continue /label "Run=2" /append {grp1} 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

```

6.213 RunAsync

RunAsync <netlist>[<datafile>]

Spawns a new simulator process and runs specified netlist.

RunAsync has the benefit over [Run](#) in that it is possible to carry on working in the front end normally while the simulation runs in the background. The disadvantage is that the asynchronous process cannot communicate with the front end. This means that incremental graph updates are not possible and the data for the simulation needs to be manually loaded after the simulation is complete.

To load the data use the [OpenGroup](#) command.

Parameters

<i>netlist</i>	Path to simulation netlist
<i>datafile</i>	Path to data file. If omitted, data will be saved to a temporary file in the temporary data directory.

6.214 RunCurrentScript

RunCurrentScript

Runs the script currently open in the text editor. The script must have been opened using the [OpenScript](#) command, created with the [NewScript](#) command or recovered from a restore session operation. The script will be run as it is displayed in the editor including any unsaved edits.

Notes

When a script is run using this command it will be referred to by the path of the file in the editor if there is one. If there isn't (i.e. the editor has never been saved, the script will be referred to as '<LocalScript>' in any error messages. This will also be the return value from the [ScriptName\(\)](#) function.

6.215 Save

Save [/all]

Saves the currently selected schematic.

Parameters

<i>/all</i>	If specified, all open schematics will be saved. Note that schematics that have not previously been saved will not be saved by this command.
<i>/nostyle</i>	If specified, the style library will not be written to the schematic.

6.216 SaveAs

SaveAs [/force] [/writeSymbol] [/tab <tabnum>] [/id id] <filename>

Saves the currently selected schematic.

Parameters

<i>/export</i>	Saves the schematic to specified file but does not change the file, if any, to which the schematic is linked. It also does not update the modified status of the schematic
<i>/force</i>	If specified and <i>filename</i> already exists it will be overwritten without prompting. Otherwise if the file exists an error will be reported.
<i>/id</i>	Use id obtained from OpenSchematic() or GetSchematicTabs()
<i>/nostyle</i>	If specified, the style library will not be written to the schematic.
<i>/tab</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() can be used to determine the number of tabs open in a window.
<i>/ui</i>	Ignored. This is retained for compatibility with version 7.2 and earlier. This was used in conjunction with <i>/tab</i> to identify a schematic from a window id and tab id. From version 8, the GetOpenSchematics() function returns all schematics that are open independent of which window, so <i>/ui</i> is no longer required
<i>/wid</i>	Use id from WM_GetContentWidgetNames
<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 not specified
<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).

6.217 SaveGraph

SaveGraph <filename>

Saves the currently selected graph to an XML file. This can subsequently be restored using [OpenGraph](#).

Parameters

<i>/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 from the function GetGraphObjects() , using <code>GetGraphObjects('graph')</code> , while GetGraphTabs() can be used to obtain the graph objects within a single window.
<i>/wid</i>	

6.218 SaveGroup

SaveGroup [/force] [/version version] [<filename>]

Saves the current group in binary format. Data groups can be opened with [OpenGroup](#)

Parameters

<i>/force</i>	If is specified, any existing file will be overwritten. Otherwise the command will fail and display an error message
<i>/version</i>	Ignored. Retained for backward compatibility.
<i>filename</i>	Save to the <i>filename</i> . Data can later be restored with the command OpenGroup . If <i>filename</i> is not specified, a dialog box will be opened allowing the user to choose from available files.

See Also

[CreateGroup](#)
[DelGroup](#)
[Groups\(\)](#)

6.219 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 notes below.

Parameters

<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.
<i>filename</i>	File where output is written.

Notes

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.

6.220 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 *User's Manual/Analysis Modes/Transient Analysis/Transient Snapshots* and *Simulator Reference Manual/Command Reference/.TRAN/Snapshots*.

6.221 SaveSymbol

```
SaveSymbol [/comp] [/file <filename>] [/lib <lib-symbol-name>] [/flags <flags>]
          <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.

Parameters

<i>/comp</i>	Saves symbol as a component to path <i>symbol-name</i> .
<i>/file</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</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 “ <i>All references to symbol automatically updated</i> ” check box in the symbol editor’s File Save... dialog box.
<i>/lib</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>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 command Inst 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.

6.222 SaveSymLib

```
SaveSymLib [/v25] [/append] [/force] /lib <libname>/all <filename>
```

Parameters

<i>/all</i>	Write out all installed symbols.
<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>/ascii</i>	Force save to ascii file format (default).
<i>/binary</i>	Force save to binary file format.
<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</i>	Name of library file to write out. This must be an installed file.
<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>filename</i>	File to receive symbols.

6.223 SaveTextEditor

SaveTextEditor

Saves the current text editor.

Parameters

/type

6.224 SaveTextEditorAs

SaveTextEditorAs [*<filename>*]

Saves the current text editor to a specific file.

Parameters

/type The text editor type to save.

6.225 SchematicEnableFileWatcher

SchematicEnableFileWatcher *<filename>*

Enabled the file watcher on the current schematic. Argument the name of the file that should be watched.

6.226 SchematicFileWatcherIgnoreChanges

SchematicFileWatcherIgnoreChanges <filename>

Disables the file watcher on a schematic editor. Argument is the filename of the schematic to disable the file watcher for.

6.227 SchematicFileWatcherWatchChanges

SchematicFileWatcherWatchChanges <filename>

Enables the file watcher on a schematic editor. Argument is the filename of the schematic to disable the file watcher for.

6.228 ScreenShotWindow

ScreenShotWindow

Captures a screen shot of the current window, saves the image to the clipboard. Resulting image will not include the window frame.

6.229 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](#).

See Also

[ScriptStep](#)

[ScriptResume](#)

[ScriptPause](#)

6.230 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](#).

See Also

[ScriptStep](#) [ScriptResume](#) [ScriptAbort](#)

6.231 ScriptResume

ScriptResume

Resumes script that has been paused with [ScriptPause](#).

See Also

[ScriptStep](#) [ScriptPause](#) [ScriptAbort](#)

6.232 ScriptStep

ScriptStep

Steps a paused script by one command. See [Debugging Scripts](#).

See Also

[ScriptAbort](#)
[ScriptPause](#)
[ScriptResume](#)

6.233 Select

Select [/wires] [/prop <prop-name>]/wire <wire-name>/all]

Select items on selected schematic. If the /prop switch is not specified the interactive select mode is entered.

Parameters

<i>/all</i>	If specified, all items on the current schematic sheet will be selected.
<i>/prop</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 value is not specified then all components possessing the property <i>name</i> will be selected.
<i>/wire</i>	Select wire with handle defined by <i>wirehandle</i> .

/wires If specified, only wires will be selected. Otherwise both components and wires 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.

6.234 SelectCurve

SelectCurve [*/unselect*] /all⟨curveId⟩

Selects/unselects the identified curve or all curves.

If the */a11* flag is used, then all curves on the currently selected graph are selected or unselected, depending on the */unselect* flag. Otherwise, a single curve must be specified with *curveId*.

Parameters

/all All curves will be selected or unselected.

/unselect Curve or curves will be unselected.

curveId Only used if */a11* flag is not used. Specifies a particular curve by its ID, which can be obtained from the functions [GetSelectedCurves\(\)](#), [GetAxisCurves\(\)](#) and [GetAllCurves\(\)](#).

6.235 SelectGraph

SelectGraph ⟨graph-id⟩

Switches the graph tabbed sheet to the graph specified by *graph-id*.

Parameters

/focuswin If specified, the window containing the specified graph will be brought into focus.

6.236 SelectLegends

SelectLegends [/unselect]

Selects or unselects all graph window legends.

Parameters

/unselect If specified, all legends are unselected. Otherwise they are selected.

6.237 SelectSchematic

SelectSchematic /id <schematic-id>| <schematic path>

Focuses on the specified schematic. Use either id or path.

Parameters

/id Specifies that the input argument is a schematic ID. This can be obtained from [OpenSchematic\(\)](#) or [GetSchematicTabs\(\)](#).

See Also

[GetSchematicTabs\(\)](#) [GetOpenSchematics\(\)](#)

6.238 SelectSimulator

SelectSimulator simulator name

Selects current simulator for selected schematic.

Parameters

simulator name Name of simulator to be selected. Current valid values are “SIMetrix” and “SIMPLIS”.

6.239 SelectSymbolPin

SelectSymbolPin <base-name>

Parameters

base-name The name of the pin to select.

6.240 SelectWireConnected

SelectWireConnected

Selects all wires connected to the currently selected elements. This will trace all wires and select all connected wires.

6.241 Set

Set [/temp] [option-spec [option-spec ...]]

Defines an option.

Parameters

/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.

/temp2

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 Chapterum:SundryTopics* of the *User's Manual*. For options of type Boolean, use form1. For others, use form 2.

6.242 SetAnnotationTextPosition

SetAnnotationTextPosition [/x <x>] [/y <y>] [/positionJustification <justification>]

Sets the position of text within a shape based annotation.

Parameters

/positionJustification Where to make position offset relative to, options are: *TopLeft, TopCenter, TopRight, MiddleLeft, MiddleCenter, MiddleRight, BottomLeft, BottomCenter* and *BottomRight*.

/x x-position of the text.

/y y-position of the text.

6.243 SetCurveName

SetCurveName <curve-id><curve-name>

Changes curves label. This is the text displayed in the legend panel.

Parameters

<i>curve-id</i>	Curve Id. Curve id is returned by the functions GetSelectedCurves() , GetAxisCurves() and GetAllCurves() .
<i>curve-label</i>	New label for curve. To restore a label to its default value set this to %DefaultLabel%.

Notes

Curve labels can also be edited using the command [SetGraphAnnoProperty](#) to edit the curve's Label property.

6.244 SetDefaultEncoding

SetDefaultEncoding <encoding>

When text files such as scripts, netlists and Verilog-A files are open in text editors and when processed, the encoding is expected to be in UTF-8 (8 bit UNICODE). The encoding affects how characters are encoded in the file. UTF-8 is a universal format that is able to render all characters world-wide but retains compatibility with 7 bit ASCII. If at least one character in an input sequence is detected that is not a valid UTF-8 sequence, an assumption has to be made as to what the encoding is. The default is to use the setting defined by the system locale which can be set in the control panel. This function can be used to set an alternative encoding.

The setting made by this command will also affect the display of text in schematics created in older versions of SIMetrix. For example, if you have a schematic created with version 7.1 on a Japanese system setup with the Japanese code-page, any Japanese characters in that schematic will not show correctly unless your system is also setup with the Japanese code page. If you execute command `SetDefaultEncoding shift-jis` this schematic will show correctly - once it is reopened. You can resave the schematic, in which case it will be saved using UTF-8 and will show correctly on any system.

The argument to the command is the encoding. Some possible values are:

- default - resets to system locale
- windows-1252 - the default on English language windows systems
- shift-jis - Japanese characters
- UTF-8 - Input unconditionally assumed to be UTF-8

A complete list of valid values is returned by the function [GetCodecNames\(\)](#). Note that the default encoding only affects behaviour when an input sequence does not comply with UTF-8. Some character encoding schemes (e.g. UTF-16) cannot be easily differentiated from UTF-8 and so are not easily detected. It is usually not appropriate to use this command to set such a default encoding.

This command sets the default encoding for the current session but will be restored to the previous encoding in future sessions. To set a persistent encoding that doesn't change, set the `DefaultEncoding` option variable as shown below:


```
Set DefaultEncoding=value
```

where value is one of the values described above. To reset to the default, use:

```
Unset DefaultEncoding
```

6.245 SetDisable

```
SetDisable <propname><propvalue>[enable|short]
```

Disables or re-enables schematic instances defined by a property name and value

Parameters

<i>property-name</i>	Property name used to identify instances to process
<i>property-value</i>	Property value used to identify instances to process
<i>If set to 'enable', instances will be enabled. If set to 'short', two-pin instances will be disabled and short-circuited. The command will fail if the instance does not have two pins. If omitted the instance will be enabled.</i>	If set to 'enable', instances will be enabled. If set to 'short', two-pin instances will be disabled and short-circuited. The command will fail if the instance does not have two pins. If omitted the instance will be enabled.

6.246 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](#).

Parameters

<i>/user</i>	If set, allows user-defined properties to be added or edited. Otherwise only built-in properties may be changed.
<i>object-id</i>	Id of object which owns the property to be edited.
<i>property-name</i>	Name of property to be edited.
<i>property-value</i>	New value of property.

6.247 SetGroup

SetGroup <group-name>

Changes the current group.

Parameters

group-name Name of new group. An array of current group names is returned by the function [Groups\(\)](#).

See Also

[Groups\(\)](#)

6.248 SetHighlight

SetHighlight /prop <propname>[<propvalue>] | /wire <wirehandle>[<colourindex>] | /net <netname>[<colour>] /all 1 | /all 0 | /clearallopen | 1 | 0

Highlights or unhighlights schematic objects.

At most one parameter switch at may be used.

Parameters

<i>/all</i>	If '1' specified, highlights all objects on selected schematic. Otherwise, unhighlights all objects on selected schematic.
<i>/clearAllOpen</i>	Clears all highlighting on all open schematics.
<i>/net</i>	
<i>/prop</i>	Property name. If specified without propvalue all instances possessing propname will be highlighted. Otherwise only instances possessing propname with propvalue will be highlighted.
<i>/wire</i>	Handle of wire to be highlighted.
<i>1 0</i>	When no switches are given, if set to '1', all selected objects highlighted, otherwise all selected objects unhighlighted.

Notes

Usage is one of the following:

1. SetHighlight /prop *propname* [*propvalue*]
2. SetHighlight /wire *wirehandle*
3. SetHighlight /all 1|0

4. SetHighlight /clearAllOpen
5. SetHighlight 1|0

6.249 SetOrigin

SetOrigin $\langle x \rangle \langle y \rangle$

Sets the origin of the current symbol.

Parameters

<i>x, y</i>	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.
-------------	--

Notes

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

[GetSymbolOrigin\(\)](#) [SetSymbolOriginVisibility](#)

6.250 SetPinPrefix

SetPinPrefix $\langle \text{pin-name} \rangle \langle \text{prefix-text} \rangle$

Sets the prefix for the selected pin property. This is a symbol editor command.

Parameters

<i>pin-name</i>	The name of the pin to change
<i>prefix-text</i>	The prefix to apply to the pin. Leave blank if no prefix is to be set.

6.251 SetPinSuffix

SetPinSuffix $\langle \text{pin-name} \rangle \langle \text{suffix-text} \rangle$

Sets the suffix for the selected pin property. This is a symbol editor command.

Parameters

<i>pin-name</i>	The name of the pin to change
<i>prefix-text</i>	The suffix to apply to the pin. Leave blank if no suffix is to be set.

6.252 SetReadOnly

SetReadOnly <vecname>

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.

6.253 SetRef

SetRef <vector-name><reference-expression>

Attaches *reference-expression* to *vector-name*. Previous reference is detached and deleted if no longer used. See [Expressions](#) for details on references.

See Also

[Expressions](#)

6.254 SetSnapGrid

SetSnapGrid <snapgrid>

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.

Parameters

<i>snapgrid</i>	Snap grid in sheet units. May be 120 (default), 60, 40, 30 or 24.
-----------------	---

6.255 SetStyleColour

SetStyleColour

Sets the style with the specified colour. The colour is specified as a hex colour code, in blue-green-red format using `0x00bbggrr`.

Parameters

<i>/colour</i>	The new colour to apply using a blue-gree-red hex code.
<i>/stylename</i>	The name of the style to change the colour for.

6.256 SetSymbolFillStyle

SetSymbolFillStyle *<style-name>*

Applies a fill style to a symbol. If symbols have a filled region, the colour of those regions can be specified to be different from the rest of the symbol.

The style name to use should be given as an argument.

Parameters

<i>style-name</i>	The name of the style to apply.
-------------------	---------------------------------

6.257 SetSymbolOriginVisibility

SetSymbolOriginVisibility *show/hide/toggle*

Controls the visibility of the origin marker in the symbol editor.

6.258 SetUnits

SetUnits *<vector-name>**<units>*

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 {''}
```

6.259 Shell

Shell [/wait] [/displayStdout] [/displayStderr] [/command] <command-string>

Launches an application.

Parameters

<i>/command</i>	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>/displayStderr</i>	
<i>/displayStdout</i>	Displays in the message window any standard output from the program.
<i>/noConnectOutPipes</i>	
<i>/wait</i>	If specified, application is launched synchronously. This means that SIMetrix will not continue until the application has closed.
<i>command-string</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 the function Shell() .

Notes

Console mode applications will be launched without the console. To run a console mode application in a manner such that the console is displayed, use the command [ShellOld](#).

6.260 ShellOld

ShellOld [/wait] [/hide] [/icon] [/command] <command-name>

Launches an application. This behaves identically to the Shell command implemented on version 4.5 and earlier.

Parameters

<i>/command</i>	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>/console</i>	
<i>/hide</i>	Start the program with the main window initially hidden.
<i>/icon</i>	Start the program in a minimised state.
<i>/wait</i>	If specified, application is launched synchronously. This means that SIMetrix will not continue until the application has closed.
<i>command-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() .

6.261 Show

Show [/file <filename>] [/append <filename>] [/noindex] [/plain] [/noHeader] [/clipboard] [/names <names>] [/force] [/width <width>] [/lock] [/detail] [/unix] expression [expression ...]

Displays the value of an expression.

Parameters

<i>/append</i>	As <i>/file</i> except that file is appended if it already exists.
<i>/clipboard</i>	If specified, the result is copied to the windows clipboard.
<i>/detail</i>	
<i>/file</i>	If specified, outputs result to <i>filename</i> . The values are output in a format compatible with OpenGroup and the <i>/text</i> switch.
<i>/force</i>	File specified by <i>/file</i> will be unconditionally overwritten if it exists.
<i>/interactive</i>	
<i>/list</i>	

<i>/lock</i>	If specified with <i>/file</i> , a lock file will be created while the write operation is being performed. The file will have the extension <i>.lck</i> . This can be used to synchronise data transfers with other applications. The file will be locked for write operations.
<i>/names</i>	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.
<i>/noHeader</i>	
<i>/noindex</i>	If the vector has no reference, the index value for each element is output if this switch is <i>not</i> specified.
<i>/plain</i>	If specified, no index (as <i>/noindex</i>), and no header (as <i>/noHeader</i>) will be output. In addition, string values will be output less the quotation marks.
<i>/unix</i>	
<i>/width</i>	Page width in columns.
<i>expression</i>	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 xvalues. 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.

6.262 ShowCurve

ShowCurve <curve-id>

Shows specified curve having been hidden using [HideCurve](#).

See Also

[HideCurve](#)

6.263 ShowSimulatorWindow

ShowSimulatorWindow

Displays simulator status window if it is currently hidden.

6.264 SizeGraph

SizeGraph [/axisid id] <x-offset><y-offset><x-scale><y-scale>

General purpose command to zoom or scroll a graph. Can apply an offset or scale or both. If both are specified, the offset is applied first.

Parameters

<i>/axisid</i>	Specify which y-axis to resize. If omitted, all y-axes on selected graph will be affected.
<i>/xfull</i>	If specified, the x-axis is zoomed to fit whole graph. <i>xscale</i> and <i>xoffset</i> will be ignored.
<i>/yfull</i>	If specified, the y-axis is zoomed to fit whole graph. <i>yscale</i> and <i>yoffset</i> will be ignored.
<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.

6.265 SwitchModelSection

SwitchModelSection /default | <section-name>

Switches model library to a new selection. This is intended to be temporary and will be restored automatically to the default in a new session. Use `SwitchModelSection /default` to restore to default setting.

Parameters

/default

6.266 TemplateEditProperty

TemplateEditProperty [/hidenew] <ref><propname><propvalue>

Edits the property of a schematic instance.

This command may only be executed in a template script. It records an instruction to edit an instance property but the instruction will not be actioned until the netlist operation has completed. So a subsequent call to `TemplateGetPropValue`, for example, will return the unedited value of the property.

Please see [Schematic Template Scripts](#) for more information. In other situations use the command [Prop](#).

Parameters

<i>/hidenew</i>	Hide added property. Does not affect any property already present on the 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

6.267 TemplateSetValue

TemplateSetValue <ref><template-value>

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.

This command may only be executed in a template script. Please see [Schematic Template Scripts](#) for more information.

Parameters

<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.

6.268 TextEditorCommentLines

TextEditorCommentLines

Comments highlighted lines in the selected text editor. Works for all text based editors and will apply the correct commenting formats for the editor type.

6.269 TextEditorFileWatcherIgnoreChanges

TextEditorFileWatcherIgnoreChanges

Disables the file watcher for current text editor. The file watcher detects when the file changes and prompts the user to reload it.

6.270 TextEditorFileWatcherWatchChanges

TextEditorFileWatcherWatchChanges

Enables the file watcher for current text editor. The file watcher detects when the file changes and prompts the user to reload it.

6.271 **TextEditorFind**

TextEditorFind

Displays the find pop-up window for the selected text editor.

6.272 **TextEditorFindNext**

TextEditorFindNext

Triggers a find next event on the current text editor. If the find dialog is not currently open, it will display the find dialog. If the dialog is open, it will trigger the find next event on that dialog.

6.273 **TextEditorGoToLine**

TextEditorGoToLine <line number>

Moves the cursor to the given line in the selected text editor.

Parameters

line number The line number to move the cursor to.

6.274 **TextEditorUncommentLines**

TextEditorUncommentLines

Uncomments highlighted lines in the selected text editor. Works for all text based editors.

6.275 **TextWin**

TextWin show/hideltoggle

Hide/Show the schematic's text window (also known as the "F11 Window") for entering simulator controls.

Using `TextWin toggle` will hide the text window if it is currently visible and vice-versa.

6.276 **Title**

Title schem|graph <title>

Changes a window's title.

Parameters

<i>schem</i>	Apply to selected schematic window.
<i>graph</i>	Apply to selected graph window.
<i>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.

6.277 TitleSchem

TitleSchem <title>

Sets the title of the current schematic.

Parameters

<i>title</i>	The new title for the schematic.
--------------	----------------------------------

6.278 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.

Parameters

<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.

Notes

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.

6.279 UndoGraphZoom

UndoGraphZoom

Restores previous graph view area. Successive execution of this command will retrace the entire history of graph magnification and scroll positions.

6.280 UngroupSelected

UngroupSelected [/all]

Ungroups selected schematic elements. If the elements are grouped several times only the most recent grouping is removed, unless the *all* flag is set.

Parameters

/all If set, all groupings are removed for the selected elements in cases where the elements are grouped several times.

6.281 UnHighlightCurves

UnHighlightCurves

Unhighlights all curves.

6.282 UnLet

UnLet <vector-name>...

Destroy vector.

Parameters

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 Let](#)

6.283 Unprotect

Unprotect

Unprotects and selects protected schematic components.

See Also

[Protect](#)

6.284 Unselect

Unselect

Unselects all components and wires on selected schematic.

Parameters

/rect

/rectmode

/wires

See Also

[Select](#)

6.285 UnSet

UnSet name [name ...]

Deletes specified option.

Parameters

/temp

Deletes only temporarily. Will revert to original value once control returns to the command line.

Notes

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.

See Also

[Set](#)

6.286 Unzip

Unzip <archive-file-path><target-directory>

Unzips a zip archive.

Parameters

<i>archive-file-path</i>	The path to the archive file to unzip.
<i>target-directory</i>	The path to target directory to unzip the content to. This will overwrite anything that already exists with the same names.

6.287 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.

6.288 UpdateAnnotationText

UpdateAnnotationText [/handle <handle>] [/text <text>]

Updates the text within the selected annotation. Uses the currently highlighted schematic editor.

Parameters

<i>/handle</i>	The handle of the annotation to update.
<i>/text</i>	The new text to apply to the annotation.

6.289 UpdateDefaultStyle

UpdateDefaultStyle /lineType [type] /lineColour [colour] /lineThickness [thickness] /fontName [name] /fontSize [size] /fontColour [colour] /italics /bold /overline /underline /propertyStyle

Updates the default global style.

Parameters

<i>/bold</i>	Bold font.
<i>/fontColour</i>	As an AABBGRRR value.
<i>/fontName</i>	Font family name.
<i>/fontSize</i>	A number.
<i>/italics</i>	Italic font.

<i>/lineColour</i>	As an AABBGRR value, 0x00ff00ff for blue=255, green=0, red=255.
<i>/lineFixedThickness</i>	Set to 1 to fix line thickness - i.e. does not scale with zoom
<i>/lineThickness</i>	Line thickness. Units 1/96 logical inches (real)
<i>/lineType</i>	Options are Solid, Dash, Dot, DashDot, DashDotDot.
<i>/overline</i>	Overline the text.
<i>/underline</i>	Underline the text.

6.290 UpdateGlobalStyle

UpdateGlobalStyle <name>/lineType [type] /lineColour [colour] /lineThickness [thickness] /fontName [name] /fontSize [size] /fontColour [colour] /italics /bold /overline /underline /propertyStyle

Updates an existing global style. Will only update the options used.

Parameters

<i>/bold</i>	110 Bold font.
<i>/fontColour</i>	As an AABBGRR value.
<i>/fontName</i>	Font family name.
<i>/fontSize</i>	Font size in points (integer).
<i>/italics</i>	110 Italic font.
<i>/lineColour</i>	As an AABBGRR value, 0x00ff00ff for blue=255, green=0, red=255.
<i>/lineFixedThickness</i>	Set to 1 to fix line thickness - i.e. does not scale with zoom
<i>/lineThickness</i>	Line thickness. Units 1/96 logical inches (real)
<i>/lineType</i>	Options are Solid, Dash, Dot, DashDot, DashDotDot.
<i>/overline</i>	110 Overline the text.
<i>/propertyStyle</i>	110 Font should appear in the Property style options drop down box.
<i>/underline</i>	110 Underline the text.
<i>/updateall</i>	Optional flag that will notify and update all Editors and update the configuration file. Use once after a batch change of style settings.
<i>name</i>	Name of the style to use.

6.291 UpdateGraphColours

UpdateGraphColours

Updates colours in all graph sheets

6.292 UpdateProperties

UpdateProperties [/all] [<property-name>[<property-value>]]

Restores properties on specified schematic instances to symbol defaults.

Command has two modes of operation. If /all is specified then all properties will be restored to the state defined in the symbol. If /all is omitted, properties that exist on the symbol but are missing on the instance will be added. All existing instance properties will be unaffected.

Parameters

<i>/all</i>	Restore all properties to symbol state. If omitted only new properties added. See description for details
<i>property-name</i>	Property name used to identify instances to process. Use selected instances if omitted
<i>property-value</i>	Property value used to identify instances to process. If omitted but <i>property-name</i> is specified, all instances with <i>property-name</i> will be processed.

6.293 UpdateRunningDialog

UpdateRunningDialog <status>

Updates a process running dialog by one step. Also allows for the status message to be updated.

See Also

[CreateRunningDialog](#)

[DestroyRunningDialog](#)

6.294 UpdateStyleInfo

UpdateStyleInfo [<style-info>]

Updates the style information for a schematic.

Input argument is a list of strings, each line defines a style in: *Name\LineType\LineThickness\LineColour*.

Parameters

<i>/fromScriptDefinitions</i>	Set this if definitions come straight from a script (fixes problem with "" around font name missing)
<i>/hierarchy</i>	
<i>style-info</i>	A list of string defining each style per string.

6.295 UpdateSymbol

UpdateSymbol <symbol-name>

Updates symbols on currently selected schematic from global symbol library.

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](#).

Parameters

symbol-name Name of the symbol to be updated.

6.296 UpdateSystemStyleInfo

UpdateSystemStyleInfo [*style-info*]

Updates the style information for a schematic at the system level. The results will be stored in the users options file.

Input argument is a list of strings, each line defines a style in: *Name\LineType\LineThickness\LineColour*.

Parameters

style-info A list of string defining each style per string.

6.297 UpdateTitleBlock

UpdateTitleBlock [/company <company name>] [/title <title name>] [/author <author name>] [/loc <x><y>]

Updates the content of a title block.

Uses the currently selected schematic editor with the selected title block.

Parameters

<i>/author</i>	Updated author name.
<i>/company</i>	Updated company name.
<i>/date</i>	Updated date string, use «auto» for assigning the date automatically when the schematic is saved.
<i>/force</i>	Use this to force selection of a title block. This will cause the first title block it comes across to be used. Use this if its possible no title block will be selected, for example operating on a file that has been opened by a script call.

<i>/layout</i>	Updated layout, either <i>Horizontal</i> or <i>Vertical</i> .
<i>/logo</i>	Updated logo image filename.
<i>/notes</i>	Updated notes. Use backslash n to mark new lines within the text.
<i>/title</i>	Updated title.
<i>/updategiven</i>	Flag that will force an update of only those elements that are defined in the command call. All other items use the values as they are currently.
<i>/version</i>	Updated version string, use «auto» for assigning the version number automatically when the schematic is saved.

6.298 UseGlobalStyles

UseGlobalStyles

Instruct current schematic to use global styles.

6.299 ViewFile

ViewFile <file name>

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.

6.300 WebOpen

WebOpen <URL>[/title <title>]

Parameters

<i>/restore</i>	Identifies this is part of a restore session call, the argument is the widget name.
<i>/title</i>	Sets the title of the View.
<i>/usesysbrowser</i>	Force system browser for all file types
<i>/welcome</i>	Identifies that this is the welcome page.
<i>URL</i>	The URL to open.

6.301 Wire

Wire [/start] [/loc <x1><y1><x2><y2>] [/mode] [/startloc <x1><y1>]

Enter schematic wiring mode.

Parameters

<i>/loc</i>	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() .
<i>/mode</i>	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.
<i>/start</i>	If specified, a new wire is started.
<i>/startloc</i>	

6.302 WireMode

WireMode On|Off

Switches schematic wiring mode on or off.

6.303 WM_CloseAllSystemWidgets

WM_CloseAllSystemWidgets

Closes all the System Views from the currently selected window.

6.304 WM_CloseNonPrimaryWindows

WM_CloseNonPrimaryWindows

This will close all windows that are not the primary window. If the window does not close, for example due to a user pressing cancel on a file modified save or not box, then this will report an error.

6.305 WM_CloseSheet

WM_CloseSheet [*<content-widget-name>*]

This will close the most recently accessed or explicitly-specified content window. A content window may be a schematic, symbol editor, graph, text editor or welcome page.

Parameters

<i>content-widget-name</i>	Name of content widget to close as returned by WM_GetContentWidgetNames
----------------------------	---

6.306 WM_ProgressWindowClose

WM_ProgressWindowClose <identifier>

Closes the specified progress window.

See Also

[WM_ProgressWindowCreate](#)

[WM_ProgressWindowCloseAll](#)

[WM_ProgressWindowReport](#)

6.307 WM_ProgressWindowCloseAll

WM_ProgressWindowCloseAll

Forces all progress windows to be closed.

6.308 WM_ProgressWindowCreate

WM_ProgressWindowCreate <number steps><identifier>[/title <window title>] [/caption <caption message>] [/message <progress message>]

Creates a progress window, with given number of steps and identifier.

The window contains a progress bar that increments each time [WM_ProgressWindowReport](#) is called.

Parameters

<i>/caption</i>	The caption of the window. This appears above the progress bar and cannot be changed after the window is created.
<i>/message</i>	The status message of the progress window. This appears below the window and can be changed after the window is created.
<i>/title</i>	The title of the window.
<i>number steps</i>	The number of times WM_ProgressWindowReport has to be called to make the progress bar be completely full.
<i>identifier</i>	The identifier that will be used to reference the progress window on update and close calls.

See Also

[WM_ProgressWindowClose](#)

[WM_ProgressWindowCloseAll](#)

[WM_ProgressWindowReport](#)

6.309 WM_ProgressWindowReport

WM_ProgressWindowReport <identifier><progress message>

Increments the progress bar and allows status message to be updated.

Parameters

<i>identifier</i>	The identifier of the progress window to update.
<i>progress message</i>	The new progress message to show in the window.

See Also

[WM_ProgressWindowClose](#)

[WM_ProgressWindowCloseAll](#)

[WM_ProgressWindowCreate](#)

6.310 WM_RevertToSaved

WM_RevertToSaved [/id <widget-id>]

Reverts a widget back to its last saved state. By default this will be the currently active widget (active window, highlighted widget).

Optional /id <widget-id> flag to select a particular widget.

Parameters

/id

6.311 WM_Undock

WM_Undock [/dock <window name>] <window-name><widget-name>

Undocks the specified content widget from the window it is in. Optionally the widget can be docked to another specified window.

Parameters

<i>/dock</i>	Optional flag, provide the name of the window to dock the widget to. If not given, the widget is docked within a new window.
<i>window-name</i>	The name of the window the widget is currently in.

widget-name The name of the widget to undock.

6.312 WriteImportedModels

WriteImportedModels [/include] <netlist><filename>

Writes all library models required by *netlist* to *filename*.

Parameters

/include

6.313 XMLAddAttribute

XMLAddAttribute <attribute-name><attribute-value><reference>

Adds an attribute to the XML at the current location.

Parameters

attribute-name The attribute name for the element being created.
attribute-value The attribute value for the element being created.
reference Reference for the XML document.

6.314 XMLAddElement

XMLAddElement <element-name><reference>

Adds an element to the XML at the current location, then sets the new element as the current focus element.

Parameters

element-name The tag name for the element being created.
reference Reference for the XML document.

6.315 XMLClose

XMLClose <reference>

Closes the XML reference.

Parameters

reference Reference for the XML document.

6.316 XMLGoUpLevel

XMLGoUpLevel <reference>

Moves the current focus element up to its parent.

Parameters

reference Reference for the XML document.

6.317 XMLNew

XMLNew <reference>

Creates a new XML reference object.

Parameters

reference Reference for the created XML document to be used to refer to it later.

6.318 XMLOpenElement

XMLOpenElement /index [idx] /tag [tag-name] <reference>

Opens the XML element and sets it as the current focus level.

Parameters

/index Chooses the element based on the index number, as defined by [XML-GetElements\(\)](#).

/tag Chooses the element based on the tag name. If there are multiple tags with the same name, it opens the first one, unless index is defined, then it uses that index position in the elements of the type requested.

reference Reference for the XML document.

6.319 XMLOpenFile

XMLOpenFile <xml-path><reference>

Opens an XML document from a file, creating a new XML reference object.

Parameters

<i>xml-path</i>	Path to the XML document to open.
<i>reference</i>	Reference for the created XML document to be used to refer to it later.

6.320 XMLSaveFile

XMLSaveFile <output-path><reference>

Writes an XML document to a file.

Parameters

<i>output-path</i>	Path to write to.
<i>reference</i>	Reference for the XML document.

6.321 Zip

Zip <archive-file-path><source-path>

Zips a file or a directory of files. If the path to a directory is given, all sub-directories are recursively zipped to the archive too.

Parameters

<i>archive-file-path</i>	The path to the archive file to create. This will overwrite anything that already exists.
<i>source-path</i>	The path to the source file or directory to recursively archive.

6.322 Zoom

Zoom rect|rectbutton|fulloutlin|<new scale>

Changes magnification of currently selected schematic.

Parameters

<i>rect</i>	Zooms to a given rectangle defined by the left, top, right and bottom position.
<i>rectbutton</i>	User selects area to be viewed with mouse.
<i>full</i>	Fits schematic to the current window.
<i>out</i>	Magnification is reduced one step.
<i>in</i>	Magnification is increased one step.

Chapter 7

Applications

7.1 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.

7.1.1 User Defined Key and Menu Definitions

Virtually the entire user interface is accessed through menus, keyboard keys or tool bar 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](#) and keys can be defined with the [DefKey](#). To define toolbars and buttons, see [Creating and Modifying Toolbars](#). Commands to define new user interface elements such as menus are usually placed in the [Startup Script](#).

Key definitions may be *context sensitive*. That is, the definition is dependent on which type of window is currently active.

7.1.2 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](#).

When editing menu.sxscr, 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.

7.1.3 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.

7.1.4 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.

7.1.5 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`. SIMetrix always searches this directory first when executing an internal script.

7.2 Custom Curve Analysis

The menus `ProbelMore 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.

7.2.1 Adding New Functions

The operations listed for the menus described above are obtained from one of two built-in text files. These files are:

<code>analysis_tree.sxscr</code>	For curve analysis functions
<code>probe_tree.sxscr</code>	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.

7.2.2 ‘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.

7.2.3 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.

7.3 Automating Simulations

7.3.1 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.

7.3.2 Running the Simulator

Simulations are started using the [Run](#) command. The Run command runs a netlist not a schematic, so you must first create the netlist using the [Netlist](#) command. 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](#) for details. If you want to test whether or a run was successful, use the [GetLastError\(\)](#) function.

7.3.3 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](#), [Select](#) and [Unselect](#). 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, 1μS pulse width and 2.5μS 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 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](#) with the switch `/plain` to write a string array to a file. The string array can be created using the function [MakeString\(\)](#) and built using custom code.

7.3.4 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](#).

7.3.5 An Advanced Example - Reading Values from a File

In this section we supply an example of quite an advanced script that 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 = GetSIMetrixFile('Text', ['open', 'all'])

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
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}
    endif
  endif
endif

```

```

        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

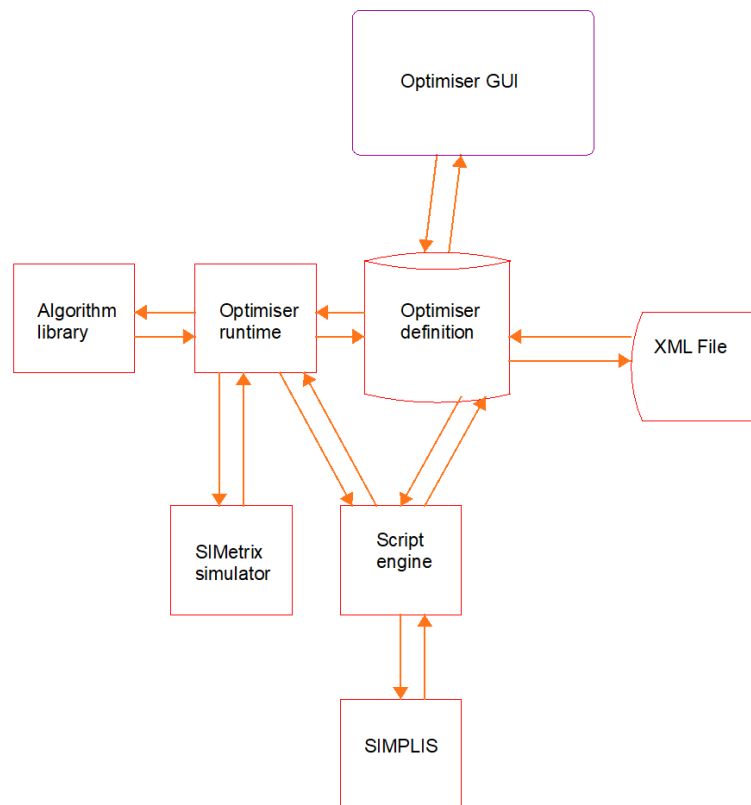
** This isn't essential, but it is always best to delete
** global variables when we are finished with them
for compIdx=0 to numComponents-1
    Let paramName = 'global:' & components[compIdx]
    UnLet {paramName}
next compIdx

```

7.4 Optimiser

The SIMetrix optimiser can be accessed using the script language and can also be used to control an optimisation process. The latter feature is used to drive SIMPLIS optimiser analyses.

The following diagram shows the structure of the optimiser framework and its interaction with the script language:



The central component is the **Optimiser definition**. This is an internal object which stores the analysis definitions, parameter definitions, measurements and other settings needed to define an optimisation session. It also stores the results of any completed optimisation.

7.4.1 Accessing Optimiser Definitions

The script language accesses the optimiser definition by obtaining an ID reference to it. The ID is an integer value which a wide range of script functions take as an argument in order to retrieve information about the optimiser or to control an optimisation session. The following functions may be used to obtain an optimiser ID:

Function Name	Description
OptimiserCreateFromXML()	Reads an optimiser definition from a file, creates an optimiser definition, then returns the ID to access it
OptimiserCreateFromXMLString()	Creates an optimiser definition from an XML string then returns the ID to access it
OptimiserWidgetCreateOptDef()	Creates an optimiser definition from the current optimiser GUI then returns an ID to it
OptimiserSimulatorGetDef()	Creates an optimiser definition from the current state of the SiMetrix simulator then returns an ID to it

7.4.2 Running an Optimiser Session

It is possible to run an optimiser session from the script language and this is how optimisation for SIMPLIS is achieved. The SIMetrix simulator is able to run optimisation sessions internally defined by netlist commands and does not usually require to be driven by the script language. But this is nevertheless possible if required. In this section the functions needed to drive an optimiser session are described.

Function Name	Description
OptimiserStart()	Starts an optimiser session
OptimiserFinish()	Terminates and optimiser session. OptimiserCloseDef() will also implicitly terminate a currently running session
OptimiserGetParameterValues()	Returns the current values for the parameters
OptimiserParameterLine()	Returns a line of text providing the names and current values of the parameters
OptimiserApplySpecification()	To be called after running all the analyses for an optimiser session. This calculates all the measurements and applies them to the optimiser session. The optimiser algorithm will then calculate new values for the parameters which can be retrieved from OptimiserGetParameterValues()

A typical sequence to run an optimiser session could be:

1. Create optimiser definition and obtain its ID
2. Start optimiser using [OptimiserStart\(\)](#)
3. Get parameter values using [OptimiserGetParameterValues\(\)](#)
4. Run all simulation analyses using parameter values obtained in step 3. If running a SIMPLIS simulation, pass the optimiser ID to the [PreProcessNetlist](#) command. This will setup the analyses and parameters for the SIMPLIS run. For SIMetrix simulations, the [OptimiserAnalysisLine\(\)](#) function returns all the analysis lines
5. Call [OptimiserApplySpecification\(\)](#)
6. Test [OptimiserRunning\(\)](#) to see if optimisation can finish. If so continue to next step. If not go back to step 3
7. Call [OptimiserClosedDef\(\)](#) to delete optimiser definition

7.4.3 Postprocessing Optimiser Results

The following functions can be used to examine the results of an optimiser session:

OptimiserResults()	Returns optimiser parameter and measurement values for best iteration of specified optimiser definition
OptimiserSimulatorResults()	Returns optimiser parameter and measurement values for best iteration of most recent SIMetrix simulation optimiser analysis
OptimiserGetIteration()	Returns optimiser parameter and measurement values for a specified iteration of specified optimiser definition
OptimiserStatus()	Returns the currents status of an optimiser session as a string

OptimiserSimulatorStatus()	Returns the current status of the SIMetrix simulation optimiser analysis as a string
OptimiserUserMessage()	Returns a message about the current state of the specified optimiser
OptimiserSimulatorUserMessage()	Returns a message about the current state of the SIMetrix simulation optimiser

7.5 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, SIMetrix includes the [MakeSymbolScript](#) command that writes a script definition of a symbol in ASCII form.

The following sections describe how to define a symbol using a script.

7.5.1 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](#) 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 PlaceFrom 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.

7.5.2 Symbol Definition Format

The following commands are used to define schematic symbols:

[AddArc](#)
[AddCirc](#)
[AddPin](#)
[AddProp](#)
[AddSeg](#)
[CreateSym](#)
[DelSym](#)
[EndSym](#)

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
```

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:

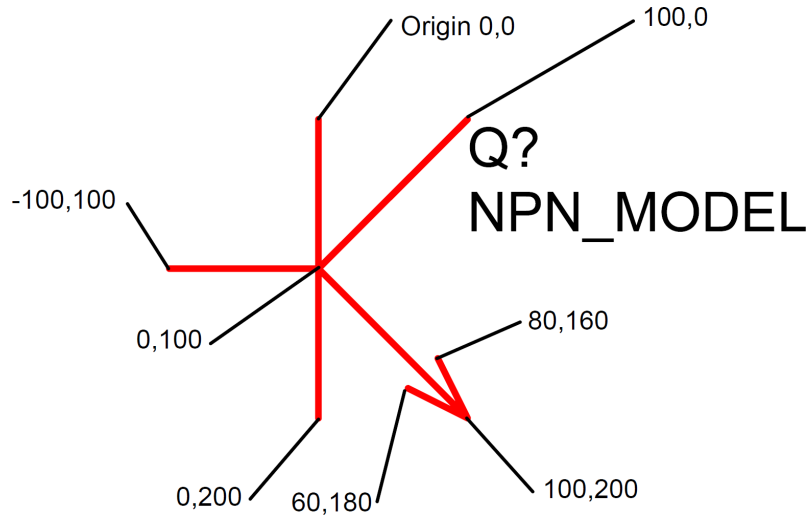
```
CreateSym npn "NPN bipolar" analog
```

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 PlaceFrom 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 *User's Manual/Further Information/Summary of Simulator Devices* for details of pin names for 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](#) 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. For more information, see [AddProp command](#).

The final line:

```
EndSym
```

terminates the model definition. The symbol will not be recognised until this is executed.

7.5.3 How Symbols are Stored

Symbol definitions are first stored in a .sxs1b 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.

7.6 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.

7.6.1 Importing Data

To import data use the `OpenGroup` command 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 `textn`, where `n` is an index as described in [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.

7.6.2 Exporting Data

To export data, use the `Show` command 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 with `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 ctrl-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 ctrl-P.

7.6.3 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.

7.6.4 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 )
1051.961874 (-5.94260987 , 0.002985269 )

```

7.7 Graph Objects

7.7.1 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.

7.7.2 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
CurveData	Curve data object. Each curve has one or more associated curve data objects. These store the curve data. Curve history is stored in curve data objects
CurveMarker	Marker used to annotate curves
FreeText	Free Text annotation object. Displays unboxed text on graph
Graph	Graph sheet
Grid	Grid object
Histogram	Histogram plot
LegendBox	Box enclosing LegendText objects
LegendText	Text objects used in legend boxes and linked to a displayed curve.
Measurement	Measurement object used to display curve measurements in the measurement window
SharedAxis	X-Axis connector object. X-Axes that are linked, that is have common limits, are linked using a SHaredAxis object
ScaterPlot	Scatter plot

Object Name	Description
SmallCursor	The small cursor that permanently follows the mouse cursor and tracks the nearest curve. There is only ever one of these on each Graph sheet
TextBox	Box enclosing FreeText object

7.7.3 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](#). 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](#).

7.7.4 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()	Add a new cursor to the currently selected graph and return its and its dimension's Ids
GetAllCurves()	Returns the IDs for all curves in the current graph
GetAllAxes()	Returns the IDs for all axes (X and Y) in current graph
GetAllXAxes()	Returns the IDs for all X-axes in current graph
GetAllYAxes()	Returns the IDs for all Y-axes in current graph
GetAxisCurves()	Returns IDs for all curves attached to specified axis (X or Y) in the current graph
GetCurrentGraph()	Returns the ID for the currently selected graph sheet
GetCursorCurve()	Returns information about curve attached to the main cursor including its ID
GetCurveAxis()	Returns ID of y-axis associated with a curve in the current graph.
GetCurveAxes()	Returns IDs for x and y axes associated with a curve in the current graph
GetDatumCurve()	Returns information about curve attached to the reference cursor including its ID
GetGraphObjects()	Returns all objects on a graph, or objects of a specified type
GetGraphTabs()	Returns ids for all graph sheets
GetGridAxes()	Returns ids for all axes belonging to the specified Grid

GetSelectedCurves()	Returns IDs of all selected curves in the current graph
GetSelectedGraphAnnotations()	Returns the ID for all selected graph annotation objects
GetSelectedGrid()	Returns ID of the currently selected Grid object in the current graph
GetSelectedXAxis()	Returns the ID of the selected X-axis in the current graph
GetSelectedYAxis()	Returns the ID of the selected Y-axis in the current graph
SelGraph()	Returns the id of the selected graph
GetGridCurves()	Returns ids of all curves in the specified Grid
CreateGraphMeasurement()	Creates a graph measurement object and returns its ID
FindGraphMeasurement()	Returns the id of a graph measurement object defined by its associated curve and its label
GetGraphObjPropValue()	function
CreateSharedAxisConnector()	Creates a SharedAxis object and returns its ID

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.

7.7.5 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.

7.7.6 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?
AutoLimit	'On' or 'Off' determines whether axis limits are adjusted automatically according to attached curves	No
AxisType	'X', 'Y' or 'Dig'	Yes
Curves	String array property returns all curve IDs attached to this axis	Yes
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
Delta	Value that determines the minor grid line spacing	No
ExpandToFitLabel	Boolean, default = 'true'. If 'true' axis size will expand to fit its label	No
Grad	Grading of axis: "Log" or "Lin".	No
Graph	ID of parent graph	Yes
Grid	ID of parent grid	Yes
GUID		Yes
ID	ID of this object	Yes
Label	Label used to annotate axis. (Actual displayed text is <label/><unit>). Default = %DefaultLabel%	No
Max	Maximum limit of axis	No
Min	Minimum limit of axis	No
Name	Axis name. ('Y1', 'Y2' etc.). Empty for X and Dig axes	Yes
ProbeId	Value used by fixed probes to identify axes	No
SharedId	ID of SharedAxis object. Only meaningful for x-axes and used to connect x-axes	No
Type	Type of object - always 'Axis'	Yes
VertOrder	Vertical order. Arbitrary string used to specify vertical display order	Yes
Unit	Physical units of axis. (E.g. 'V', 'A' etc.). Default = %DefaultUnit%	No

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

Name	Description	Read Only?
Point	1 = Reference cursor. 2 = Main 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
Colour	RGB colour value, or option setting that holds an RGB value. Default=GraphColourCrossHair (option value)	No
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	Boolean. Set to 'true' to hide cursor	No
Label	Cursor label displayed at base	No
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 dotdash style	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 affect nonmonotonic 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
GUID		Yes
ID	ID of this object	Yes
Type	Type of object - always 'CrosshairDimension'	Yes
Vertical	0 = Horizontal dimension, 1 = Vertical dimension	Yes
X1	Horizontal type only. Holds the value of the x data position of the first crosshair.	Yes

Name	Description	Read Only?
X2	Horizontal type only. Holds the value of the x data position of the second crosshair	Yes
Y1	Vertical types only. Holds the value of the y data position of the first crosshair	Yes
Y2	Vertical types only. Holds the value of the y data position of the second crosshair and is readonly	Yes
XDiff	= X2-X1	Yes
YDiff	= Y2-Y1	Yes
Crosshair1	ID of crosshair 1	No
Crosshair2	ID of crosshair 2	No
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	Boolean. If 'true', dimension object will be 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
Style	Controls display of dimension labels. Possible values: Internal Show difference only (label3) - internal position External Show difference only (label3) - external position Auto Show difference only (label3), position chosen automatically P2P1 Show absolute labels (label1 and label2) P2P1Auto Show all labels. Difference position chosen automatically None No controls selected	No

Curve, Histograms and Scatter Plots

The following properties are common to Curves, Histograms and Scatter Plots. In the following properties table, the object is referred to as a Curve but definitions apply equally to Histograms and Scatter Plots.

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
Expression	Expression that created this curve	Yes
Graph	ID of parent graph	Yes
Grid	ID of parent Grid	Yes
GroupName	The data group that was current when the curve was created	Yes

Name	Description	Read Only?
ID	ID of this object	Yes
Limits	The X and Y limits of the curve in the form [xmin, xmax, ymin, ymax]	Yes
MeasurementIds	String array. List of Measurement object IDs associated with this curve object	Yes
Measurements	Measurements added to a curve	Yes
MeasureSpec	String defining measurements for the curve. This is set for curves created by fixed probes. (.GRAPH statements)	Yes
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
ShortLabel	A label composed from Name and Suffix but without group name	Yes
SweepType	For grouped curves, defines the type of sweep that created the group. Values are 'Statistical' for Monte Carlo analyses, 'Stepped' for stepped analyses or 'None' for non-grouped curves. F	Yes
Type	Type of object - 'Curve', 'Histogram' or 'ScatterPlot'	Yes
UserProbeID	Used by measurements for fixed probes (.GRAPH statements)	Yes
XPhysType	Physical type of curves x-values	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
XY	Boolean value. 'true' if curve created from X-Y data	Yes
YAxis	ID of y-axis attached to curve	Yes
YPhysType	Physical type of curves y-values	Yes
YUnit	Physical type of curve's y-data	Yes
Hidden	Boolean. If 'true' curve will not be visible. (Inverse of Visible property)	No
Highlight	Boolean. Curve is highlighted if 'true'.	No
HighlightedDivisions	Controls selective highlighting of grouped curves. Stores a list of divisions that are highlighted in form [div1, div2, ...]. E.g. [1,5,7] would highlight divisions 1 5 and 7	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
ProbeGUID	GUID originating from probe that created curve. If the curve is created from a .GRAPH statement in the netlist, this will be the value of the GUID parameter. If created using a Curve or Plot script command this can be set using the /guid switch. This feature is currently used to implement auto probe colouring	Yes

Name	Description	Read Only?
ProbeInst	Used by fixed probes (.GRAPH statement) to manage curve history	No
RGBColour	Colour of curve. Can be entered as value returned from GetColourSpec() or using format #rrggbb where rr, gg and bb are two digit hex values representing the red, green and blue colour content respectively	No
Selected	Boolean - holds selected state of curve	No
Sequence	Integer value that is used to define default colour. The actual colour used may be set globally using options dialog box	No
Suffix	This is assigned when the result of a multistep 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
UseSequenceColour		
Visible	If 'false', curve will be hidden, but its legend display will remain	No

Curve Only

The following properties apply to Curves only, i.e they do not apply to Histograms or Scatter Plots.

Name	Description	Read Only?
Alpha	Alpha blend ratio as a percentage. 0=invisible, 100=fully opaque	No
CurveDatas	String array. List of curve data objects associated with this Curve. Note that this returns all CurveData objects available, not just the ones that are visible	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	No
ForceFastRender	Boolean. If set, unconditionally inhibits use of anti-alias with pixel width greater than 1. This combination renders slowly and is automatically disabled for large curves	No
LineStyle	Controls line style. May have values as defined below 'auto' Solid for main curve. History curves are set to broken lines in order dot, dash, dot-dash 'solid' solid 'dot' dotted 'dash' dashed 'dotdash' dot-dash	No
Persistence	Integer. History depth display. Previous curve content may be displayed up to the depth defined by this property	No

Name	Description	Read Only?
ShowPoints	If 'true' data point markers will be displayed	No

CurveData

CurveData objects store the data associated with a curve. A curve can have more than one curve data object associated with it.

Name	Description	Read Only?
Curve	ID of Curve associated with this CurveData object	Yes
Graph	Parent Graph ID	Yes
GUID		Yes
ID	ID of this object	Yes
Type	Type of this object - always CurveData	Yes
LineStyle		No
	'auto' Style chosen according to position in history	
	'solid' solid	
	'dot' dotted	
	'dash' dashed	
	'dotdash' dot-dash	

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
GUID	GUID	Yes
ID	ID of this object	Yes
Type	Type of object - always 'CurveMarker'	Yes
Colour	Background colour	No
Curve	ID of attached curve	No
Font	Font for label	No
FontColour	Text colour	No
Hidden	Boolean. If 'true' curve marker will be hidden	No
Label	Label of curve marker. May be a symbolic value. Default is %curve:label% which returns the label of the attached curve	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

Name	Description	Read Only?
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
GUID	GUID	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

Name	Description	Read Only?
Alignment	String to define alignment: 'LeftTop' 'LeftBottom' 'LeftMiddle' 'LeftAuto' 'RightTop' 'RightBottom' 'RightMiddle' 'RightAuto' 'CentreTop' 'CentreBottom' 'CentreMiddle' 'CentreAuto' 'AutoTop' 'AutoBottom' 'AutoMiddle' 'AutoAuto' The 'Auto' alignment options align according to the position of the object in the grid. X1=0 aligns to left, X1=1 aligns to right, Y1=0 aligns to bottom, Y1=1 aligns to top. The Centre values may use the alternative Center spelling.	No
Font	Font for label	No
FontColour	Text colour	No
GlobalPosition	Boolean. If true, the position defined by X1 and Y1 will be relative to the entire graph drawing area. Otherwise it is relative to the Grid specified by the Grid property	No
Grid	ID for grid used for local position	No
Hidden	Boolean. If 'true', object will be hidden	No
Label	Text displayed. Symbolic values may be used. E.g. %Time% will display the time the object was created.	No
X1	X location of object in view units. The Alignment, Grid and GlobalPosition affect the interpretation of this value	No
Y1	Y location of object in view units. The Alignment, Grid and GlobalPosition affect the interpretation of this value	No

Graph

Name	Description	Read Only?
DefaultRef	The name of the reference vector (x-values) for the first curve added to the graph. Typically this would be 'Time' or 'Frequency' for typical transient and AC analyses respectively. The property is used to manage shared x-axes	Yes
DefaultSharedAxis	ID of the SharedAxis object used to connect x-axes. There can be any number of SharedAxis objects but there is always a single SharedAxis object that new x-axes are connected to. This may be read from this property	Yes
FirstCurve	ID of the oldest curve on the graph	Yes
Graph	ID of this object	Yes
Grids	String array. IDs of all grids in the graph	Yes
GroupTitle	Title of the data group that was current when the graph was created	Yes
GUID	GUID	Yes
ID	ID of this object	Yes
SmallCursor	ID of the SmallCursor object	Yes
SourceGroup	The data group that was current when the graph was created	Yes
Type	Type of object - always 'Graph'	Yes
AcceptNewProbes	Boolean. If 'true', graph will accept curves created by fixed probes even if it was initially used for random probes. Usually fixed probes create a new graph sheet on the first run	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 graph and status bar	No
MainCursor	ID of Crosshair object comprising the main cursor. Value = -1 if cursors are not enabled.	Yes
Path	Path of file to save to. This is the file that will be used by the "FileSave" menu. When saving a graph, this property will be set to the full path name of the file. When a graph file is loaded, this property is set to the full path of the input file. (Note that this does not apply to graphs converted from the old binary format)	Yes
ProbeId	Used to manage fixed probes	No
RefCursor	ID of Crosshair object comprising the reference cursor. Value = -1 if cursors are not enabled.	Yes
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
Visible	Boolean value set to TRUE if the graph is currently visible or partially visible. This property will be FALSE if the graph window is iconised or obscured by other windows	Yes

Grid

Grids are where curves are displayed and have at least one y-axis and one x-axis. There may be one or more grids in a graph.

Name	Description	Read Only?
Curves	String array.	Yes
Graph	Parent graph ID	Yes
GUID	GUID	Yes
ID	ID of this object	Yes
ProbeId	Used to manage fixed probes	Yes
SelectedYAxis	ID of the selected y-axis	Yes
Type	Object type. Always set to 'Grid'	Yes
XAxes	String array. IDs of all x-axes attached to the grid	Yes
YAxes	String array. IDs of all y-axes attached to the grid	Yes
GridType	'Norm' (normal) or 'Dig' (digital). Digital axes are small and designed for digital traces, although they can be used for any curve	No
VertOrder	Integer. Defines vertical position of grid. For normal (non-digital) grids the grids are displayed in order bottom to top - the higher the value of VertOrder the higher up the display. The order for digital grids is the reverse of this	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.

Name	Description	Read Only?
Graph	ID of parent graph	Yes
GUID	GUID	Yes
ID	ID	Yes
Objects	Semicolon delimited string providing the IDs for each of the LegendText objects defining text entries in the box	Yes
Type	Type of object. Always 'LegendBox'	Yes
Alignment	See Alignment property of FreeText object (FreeText)	No
AutoWidth	'On' or 'Off'. If 'On' the width and height of the box automatically adjust according to contents. Otherwise the X2 and Y2 values may be adjusted to set the box dimensions	No
Colour	Background colour	No
Font	Text font	No
FontColour	Text colour	No

Name	Description	Read Only?
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 LegendText object that displays the text. The default value is %DefaultLabel%	No
Hidden	Boolean. If 'true' object will be hidden	No
X1	X position in view units	No
X2	Width of box in mm if AutoWidth set to off	No
Y1	Y position of box in view units	No
Y2	Height of box in mm if AutoWidth set to off	No

LegendText

LegendText objects are used to load legend boxes and cannot be instantiated independently.

Name	Description	Read Only?
Curve	ID of attached curve	Yes
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
DefaultLabel	The default value for the label. Actually equivalent to %Curve:Label%\n%Curve:Measurements%. ((n)denotes a new line).	Yes
Division	For grouped multi-step curves there is a LegendText object for each division. The division index may be read from this property	Yes
Graph	ID of parent graph	Yes
GUID	GUID	Yes
ID	ID of this object	Yes
Parent	ID of parent legend box	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 'LegendText'	Yes
Version	Product name and version	Yes
Font		No
Label	Set to the value of the legend box's LegendLabel property when it is added to the box	No

Measurement

Measurement objects are displayed in the measurement window and are created when a measurement is made on a curve. The display in the measurement window has three columns showing the label of the associated curve, the measurement name and the measurement value.

For measurements made on multi-division data generated by multi-step runs, the measurement object has child objects that display additional data relevant for the multi-step analysis.

Name	Description	Read Only?
Curve	ID of associated curve	Yes
DivisionLabel	String array. If Expression evaluates to a multi-division vector, this property will hold its division labels. Typically these are the labels associated with each step in a multi-step run. For example, if a resistor is swept, each division label will be in the form 'Rxx=value'	Yes
Graph	ID of parent Graph	Yes
GUID	GUID	Yes
ID	ID of this object	Yes
MultiDivLabels	String array. Multi division labels displayed in the measurement window. This depends on the value of the MultiDivMode property. See MultiDivMode property for further details	Yes
MultiDivValues	String array. Multi division values displayed in the measurement window. This depends on the value of the MultiDivMode property. See MultiDivMode property for further details	Yes
NumDivisions	Number of divisions in the evaluated Expression	Yes
RawValues	String array. Raw result of evaluating Expression without any formatting or statistical collation	Yes
Statistics	String array. Statistics values	Yes
Statistics_Names	String array. Same size as Statistics property. Parameter names for statistics values	Yes
Type	Object type. Always set to 'Measurement'	Yes
Value	String Array. Similar to RawValues but with the Template applied	Yes
Expression	Expression to be evaluated. Typically this would refer to the data from the associated curve using the expression 'cv(%curve%)' but this is not compulsory.	No
Label	Label displayed in the Label column	No

Name	Description	Read Only?								
MultiDivMode	<p>Mode for the display of multi-division data. Multi-division data is generated by multi-step runs such as parameter sweeps and Monte Carlo runs. For such runs there are two ways to display the measurement data: multiple and statistical. In multiple mode, an individual measurement is displayed for each division in the data; divisions correspond to step in a multi-step run. In statistical mode, statistical information is calculated such as mean, standard deviation, minimum and maximum.</p> <p>The valid values for this property are</p> <table> <tr> <td>Curve</td> <td>The mode (multiple or statistical) is obtained from the curve data itself</td> </tr> <tr> <td>Multiple</td> <td>multiple mode</td> </tr> <tr> <td>Statistics</td> <td>statistical mode</td> </tr> <tr> <td>AdvancedStatistics</td> <td>statistical mode - additional statistical values are included. Currently this includes Skewness and Kurtosis</td> </tr> </table> <p>This property controls the data that is actually displayed in the measurement window.</p>	Curve	The mode (multiple or statistical) is obtained from the curve data itself	Multiple	multiple mode	Statistics	statistical mode	AdvancedStatistics	statistical mode - additional statistical values are included. Currently this includes Skewness and Kurtosis	No
Curve	The mode (multiple or statistical) is obtained from the curve data itself									
Multiple	multiple mode									
Statistics	statistical mode									
AdvancedStatistics	statistical mode - additional statistical values are included. Currently this includes Skewness and Kurtosis									
Template	<p>Template applied to raw values to obtain the displayed values. This can be any text string including special values as shown below</p> <table> <tr> <td><code>%yn%</code></td> <td>Where <i>n</i> is any integer 1 or higher. <i>y1</i> returns the y-value result of the expression. If the expression evaluates to an array, <i>y2</i> returns the second element, <i>y3</i> the third etc.</td> </tr> <tr> <td><code>%xn%</code></td> <td>As <code>%yn%</code> but returns <i>x</i> values</td> </tr> <tr> <td><code>%uy1%</code></td> <td>The units for the y-values</td> </tr> <tr> <td><code>%ux1%</code></td> <td>The units for the x-values</td> </tr> </table>	<code>%yn%</code>	Where <i>n</i> is any integer 1 or higher. <i>y1</i> returns the y-value result of the expression. If the expression evaluates to an array, <i>y2</i> returns the second element, <i>y3</i> the third etc.	<code>%xn%</code>	As <code>%yn%</code> but returns <i>x</i> values	<code>%uy1%</code>	The units for the y-values	<code>%ux1%</code>	The units for the x-values	No
<code>%yn%</code>	Where <i>n</i> is any integer 1 or higher. <i>y1</i> returns the y-value result of the expression. If the expression evaluates to an array, <i>y2</i> returns the second element, <i>y3</i> the third etc.									
<code>%xn%</code>	As <code>%yn%</code> but returns <i>x</i> values									
<code>%uy1%</code>	The units for the y-values									
<code>%ux1%</code>	The units for the x-values									

SharedAxis

The SharedAxis object is used to connect x-axes. X-axes on the same graph are initially connected so that they share most of their properties, in particular their limits. So when one x-axis is zoomed, other connected x-axes change their zoom accordingly.

X-Axes always point to a SharedAxis object through their SharedId property. X-Axis that share the same SharedId will be connected.

Name	Description	Read Only?
ConnectedAxes	String array. List of axes connected to this object	Yes
Graph	ID of parent graph object	Yes
GUID	GUID	Yes
ID	ID	Yes
Type	Type of this object. Always 'SharedAxis'	Yes

SmallCursor

The SmallCursor is a cursor that follows the mouse cursor but tracks displayed curves providing an X and Y readout of the curve's data. There is only ever one SmallCursor object in each graph.

Name	Description	Read Only?
Curve	ID of curve that the cursor is currently tracking	Yes
Division	Division index of curve that the cursor is currently tracking	Yes
Graph	ID of parent graph	Yes
GUID	GUID	Yes
ID	ID of this object	Yes
Type	Type of object. Always 'SmallCursor'	Yes
X1	Current x-position	Yes
Y1	Current y-position	Yes
Hidden	Boolean. If 'true' cursor will not be visible	No

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
GUID	GUID	Yes
ID	ID of this object	Yes
Type	Type of object - always 'TextBox'	Yes
Alignment	See Alignment property of FreeText object (FreeText)	No
Colour	Background colour. Either the name of a colour object or a colour specification.	No
Font	Font used for text objects added to the box. In practice this only affects the LegendBox object which is based on TextBox.	No
FontColour	Colour used for text. Either the name of a colour object or a colour specification	No
Grid	ID of Grid where object is located. Position values X1 and Y1 are relative to this grid	No
Hidden	Boolean. If 'true' object will be hidden	No
Label	Text displayed in box	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	Yes
Y2	Physical height of box in mm	Yes

7.7.7 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 coordinate 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.

7.8 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:

<code>on_graph_anno_doubleclick</code>	Called when the user double clicks on certain graph objects
<code>on_accept_file_drop</code>	Called when a file or directory is dropped on a SIMetrix window.
<code>on_schem_double_click</code>	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](#)). Details on these event scripts follow.

7.8.1 `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.

7.8.2 `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.

7.8.3 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.

7.9 User Defined Script Based Functions

7.9.1 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 script "reg_user_funcs". The source for these can be found on the installation CD.

7.9.2 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.

7.9.3 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-num-args] [max-num-args]
```

For details see [RegisterUserFunction](#).

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.

7.9.4 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
```

7.10 User Defined Binary Functions

7.10.1 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.

7.10.2 Documentation

Documentation and associated header and example files are provided on the install CD. See the directory CD/Script/user-function-interface.

7.11 Non-interactive and Customised Printing

7.11.1 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](#)
- [NewPrinterPage](#)
- [OpenPrinter](#)
- [PrintGraph](#)
- [PrintSchematic](#)

The functions are:

- [GenPrintDialog](#) (for interactive printing)
- [GetPrinterInfo](#)

Each of these commands and functions is described in detail in its relevant section. Here we give a general overview for the printing procedure.

7.11.2 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.

7.11.3 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 GetPrinterInfo
** 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.

7.12 Schematic Template Scripts

7.12.1 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.

7.12.2 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.

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.

7.12.3 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](#) command.

7.12.4 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.

7.12.5 Template Commands and Functions

This 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 proptype propvalue

Edits a property's value. Like `TemplateGetPropValue` it is much faster than the regular commands but only works in a template script. Note that this command records an instruction to edit a property's value but the instruction will not be actioned until the netlist operation has completed.

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.

7.13 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.

7.13.1 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` to reassign the buttons. The following table shows the name of the 'Set' variable to use for each one.

'Set' Variable Name	Toolbar
ComponentButtons	SimetrixSchemToolBar
SIMPLISComponentButtons	SimplisSchemToolBar
SchematicMainButtons	SimetrixSchemToolBar and SimplisSchemToolBar
SchematicFileButtons	SimetrixSchemToolBar
SchematicFileButtonsSIMPLIS	SimplisSchemToolBar
SchematicProbeButtons	SimetrixSchemToolBar and SimplisSchemToolBar

'Set' Variable Name	Toolbar
GraphMainButtons	GraphMain
SymbolFileButtons	SymbolMain
SymbolMainButtons	SymbolMain

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 [Pre-defined Buttons](#).

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 tool bar.

You can then use the [Set](#) command to add a new button.

You can also use [UnSet](#) command 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.

Any changes made using the [Set](#) command will not take place until you restart SIMetrix/SIMPLIS. If desired you can force a rebuild of the toolbar for each window type by executing the following built-in scripts:

```
Schematic      update_schematic_toolbar
Symbol editor  update_symbol_toolbar
Graph         update_waveform_toolbar
```

7.13.2 Redefining Button Commands

You can change the command executed when a button is pressed using the command [DefButton](#). 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 [Pre-defined Buttons](#) for a complete list.

7.13.3 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](#). 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.

You can make your graphic any size, but we recommend using 64x64.

When you have created your image, you should save or copy it to the images directory. This is located at `simetrix-root/support/images`, where `simetrix-root` is the top level directory in the SIMetrix tree.

2. Execute the command `CreateToolButton`. 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](#) 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`. Again, this should be place in your startup script.
4. Add the button to a toolbar. See [Modifying Existing Toolbars and Buttons](#) 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](#).

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.

Its 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](#) for more details.

7.13.4 Creating New Toolbars

To create a completely new toolbar, use the command `CreateToolBar`. This will create an empty toolbar.

To add buttons to a new toolbar, you must use the command `DefineToolBar`. You can add both pre-defined and user-defined buttons to a custom toolbar.

7.13.5 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.

Button name	Description	Bitmap
AsciiFileEditorClose	Close Ascii File	close.png
AsciiFileEditorNew	New SIMetrix Schematic	filenew.png
AsciiFileEditorOpen	Open Ascii File	open.png
AsciiFileEditorSave	Save Ascii File	save.png
BiasV	Place Bias Marker	biasv.png
CalcAveragePower	Display Average Power/Cycle	avg.png
CalcFall	Display Fall Time	falltime.png
CalcHighPass3db	Display -3dB Point (High Pass)	3dbhighpass.png
CalcLowPass3db	Display -3dB Point (Low Pass)	3dblowpass.png
CalcRise	Display Rise Time	risetime.png
CalcRMS	Display RMS/Cycle	rms.png
Capacitor	Place Capacitor	capacitor.png
Copy	Duplicate	copy.png
Delete	Cut	cut.png
DeleteAllCurves	Delete All Curves	deleteall.png
DeleteAxis	Delete Axis/Grid	delgrid.png
DeleteCurve	Delete Curve	delete.png
Detach	Detach	detach.png
Diode	Place Diode	diode.png
Flip	Flip	flip.png
GraphClose	Close Graph	fileclose.bmp
GraphNew	New Graph	filenew.png
GraphNewCurve	Add Curve	newcurve.png
GraphNewFourier	Add Fourier Spectrum Plot	newfourier.png
GraphOpen	Open Graph	fileopen.bmp
GraphSave	Save Graph	filesave.bmp
Ground	Place Ground	ground.png
HideCurves	Hide Selected Curves	hide.png
IGBT	Place IGBT	igbt.png
Inductor	Place Inductor	inductor.png
IProbe	Place Current Probe	iprobeplaced.png
IProbeInteractive	Probe Pin Current (interactive)	iprobe.png
ISource	Place Current Source	isource.png
LogicDefinitionEditorClose	Close Logic Definition	close.png
LogicDefinitionEditorNew	New SIMetrix Schematic	filenew.png
LogicDefinitionEditorOpen	Open Logic Definition	open.png
LogicDefinitionEditorSave	Save Logic Definition	save.png
Mirror	Mirror	mirror.png

Button name	Description	Bitmap
MoveCurve	Move Curve to Selected Axis/Grid	movecurve.png
NetlistEditorClose	Close Netlist	close.png
NetlistEditorNew	New SIMetrix Schematic	filenew.png
NetlistEditorOpen	Open Netlist	open.png
NetlistEditorSave	Save Netlist	save.png
NetlistRun	Run SIMetrix Netlist	run.png
NewAxis	New Axis	newaxis.png
NewCurve	Add Curve	newcurve.png
NewFourier	Add Fourier Spectrum Plot	newfourier.png
NewGrid	New Grid	newgrid.png
NJFET	Place N-channel JFET	njfet.png
NMOS3IC	Place 3 term N-channel MOSFET	nmos_ic3.png
NMOS4	Place 4 term N-channel MOSFET	nmos_ic.png
NMOS	Place N-channel MOSFET	nmos.png
NPN	Place NPN Transistor	npn.png
Opamp	Place Opamp	opamp.png
PJFET	Place P-channel JFET	pjfet.png
PMOS3IC	Place 3 term P-channel MOSFET	pmos_ic3.png
PMOS4	Place 4 term P-channel MOSFET	pmos_ic.png
PMOS	Place P-channel MOSFET	pmos.png
PNP	Place PNP Transistor	pnp.png
Print	Print	print.png
PSU	Place PSU	psu.png
Redo	Redo	redo.png
Resistor	Place Resistor (Box shape)	resistor.png
ResistorZ	Place Resistor (Z shape)	resz.png
Rotate	Rotate	rotate.png
SatInd	Place Saturable Inductor	sat_ind.png
SatTx	Place Saturable Transformer	tx_sat.png
SchemClose	Close Schematic	close.png
SchemFind	Search	find.png
SchemNew	New SIMetrix Schematic	filenew.png
SchemOpen	Open Schematic	open.png
SchemSave	Save Schematic	save.png
SchemSaveAll	Save All Schematics	saveall.png
SCR	Place Thyristor	scr.png
ScriptEditorClose	Close Script	close.png
ScriptEditorNew	New SIMetrix Schematic	filenew.png
ScriptEditorOpen	Open Script	open.png
ScriptEditorSave	Save Script	save.png

Button name	Description	Bitmap
ScriptRun	Run Script	run.png
ShellNewSchem	New Schematic	filenew.png
ShowCurves	Show Selected Curves	show.png
SIMatrixOptions	Options	options.png
SimPause	Pause Simulation	pause.png
SIMPLISOptions	SIMPLIS Options	simplis_options2.png
SimRunSchem	Run Schematic	run.png
SymbolClose	Close Symbol	close.png
SymbolCut	Cut Selected	cut.png
SymbolDetach	Detach Selected	detach.png
SymbolDuplicate	Duplicate Selected	copy.png
SymbolFlip	Flip Selected	flip.png
SymbolMirror	Mirror Selected	mirror.png
SymbolNew	New Symbol	filenew.png
SymbolOpen	Open Symbol	open.png
SymbolRedo	Redo	redo.png
SymbolRotate	Rotate Selected	rotate.png
SymbolSave	Save Symbol	save.png
SymbolUndo	Undo	undo.png
SymbolWireMode	Segment Mode	pencil.png
SymbolZoomBox	Zoom to Rectangle	zoomrect.png
SymbolZoomIn	Zoom In	zoomin2.png
SymbolZoomOut	Zoom Out	zoomout2.png
TextEditorClose	Close Text	close.png
TextEditorNew	New SIMatrix Schematic	filenew.png
TextEditorOpen	Open Text	open.png
TextEditorSave	Save Text	save.png
TitleCurve	Change Curve Name	curvetitle.png
TL	Place Transmission Line	tl.png
Tx	Place Transformer	tx.png
Undo	Undo	undo.png
UndoZoom	Undo Zoom	undo.png
VerilogAEditorClose	Close Verilog-A	close.png
VerilogAEditorNew	New SIMatrix Schematic	filenew.png
VerilogAEditorOpen	Open Verilog-A	open.png
VerilogAEditorSave	Save Verilog-A	save.png
VerilogHDLEditorClose	Close Verilog HDL	close.png
VerilogHDLEditorNew	New SIMatrix Schematic	filenew.png
VerilogHDLEditorOpen	Open Verilog HDL	open.png
VerilogHDLEditorSave	Save Verilog HDL	save.png

Button name	Description	Bitmap
VProbe	Place Voltage Probe	vprobeplaced.png
VProbeInteractive	Probe Voltage (interactive)	vprobe.png
VSource	Place Voltage Source	vsource.png
Waveform	Place Waveform Generator	vsig.png
WebBack	Back	back.png
WebForward	Forward	forward.png
WebNew	New SIMetrix Schematic	filenew.png
WebReload	Reload	reload.png
WebSIMetrixOptions		options.png
WebSIMPLISOptions		simplis_options2.png
WebStop	Stop	stop.png
Wire	Wire Mode	pencil.png
Zener	Place Zener Diode	zener.png
ZoomFull	Zoom to Fit	zoomfull.png
ZoomFullGraph	Fit Window	zoomfull.png
ZoomIn	Zoom In	zoomin2.png
ZoomOut	Zoom Out	zoomout2.png
ZoomRect	Zoom Box	zoomrect.png
ZoomXAuto	Fit Width	zoomwidth.png
ZoomYAuto	Fit Height	zoomheight.png

7.14 Custom Dialog Boxes

7.14.1 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 “Designer” supplied with SIMetrix. “Designer” is redistributable tool provided with the Qt toolkit used for SIMetrix UI development.

7.14.2 Starting “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 “bin64” folder under the SIMetrix root. The SIMetrix installer does not create a short cut to this but you may create one yourself if required.

7.14.3 Developing Dialogs

The basic procedure is:

1. Start Designer
2. Select **New...** then choose either Dialog with Buttons Bottom or Dialog with Buttons Right. Click on Create

3. Set the top level object's `objectName` property to the required name of the SIMetrix script function. To begin with this is 'Dialog' and has the `QDialog` class name. Change 'Dialog' to a name of your choice. The name must not clash with a name already used by another dialog or a standard SIMetrix script function
4. Edit `windowTitle` property as required. This is the text that will appear in the caption bar of the dialog
5. Add widgets as required. See next section for further details. See also [Using Geometry Management](#).
6. Save result as a `.ui` file to the directory `support/dialogs` under SIMetrix root (Windows). 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 `"%SXAPPDATAPATH%/userdialogs"` resolves to a directory under the application data path. Note that you must restart SIMetrix after changing the path.

7.14.4 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 32 arguments so this property may not be larger than 31
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.

The EditBox widget may be configured to accept signal data pasted from a schematic or graph window. This is the same function as provided with the Define Curve Dialog box opened with the schematic menu **Probe | Define Curve...** and others. When you click on a node in the schematic, its node name is pasted directly in the edit box. To enable this functionality with the EditBox widget, set the `enablePasteText` property to `true`.

When `enablePasteText` for at least one EditBox widget is set, the whole dialog box will be opened in a 'Modeless' fashion. This means that it is possible to switch to other windows while the dialog is open.

TextEdit

As EditBox but multi-line and does not support the `enablePasteText` property.

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

Property Name	Description
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 'clicked' if the button was clicked to close the box or 'notclicked'.

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.

7.14.5 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 toolbar and also with the context popup menu in the form and the object inspector. 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 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.qt.io>.

See menu **Help | About...** for the current version.

7.14.6 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. 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','1'], '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')
```

7.14.7 ExecuteDialog Function

The ExecuteDialog function executes a .ui file directly using the dialog definition's full path name. The first argument to this function is the full path to the dialog .ui 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 .ui 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 32 arguments. 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 31. If calling the dialog directly, the limit is 32.

7.14.8 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.

7.15 Pre-defined Buttons

This is a list of predefined buttons that can be used with [DefineToolBar](#) for creating custom toolbars.

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
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 Nchannel MOSFET
NMOS4	nmos_ic.bmp	Place 4 term Nchannel MOSFET

Button Name	Graphic	Function
NPN	nnp.bmp	Place NPN Transistor
Opamp	opamp.bmp	Place Opamp
Options	options.bmp	Options
PJFET	pjfet.bmp	Place P-channel JFET
PMOS	pmos.bmp	Place P-channel MOSFET
PMOS3IC	pmos_ic3.bmp	Place 3 term Pchannel MOSFET
PMOS4	pmos_ic.bmp	Place 4 term Pchannel MOSFET
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
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

Button Name	Graphic	Function
ZoomXAuto	zoomwidth.bmp	Fit Width
ZoomYAuto	zoomheight.bmp	Fit Height

