

Reference Manual

Generated by Doxygen 1.7.1

Mon Apr 18 2011 11:37:58

Contents

1	TinkerCell Core Library	1
2	Module Index	9
2.1	Modules	9
3	Class Index	11
3.1	Class Hierarchy	11
4	Class Index	13
4.1	Class List	13
5	Module Documentation	17
5.1	TinkerCell Core classes	17
5.1.1	Detailed Description	20
5.1.2	Function Documentation	20
5.1.2.1	cloneGraphicsItem	20
5.1.2.2	cloneGraphicsItems	20
5.1.2.3	getGraphicsItem	20
5.1.2.4	getHandle	21
5.1.2.5	getHandle	21
5.1.2.6	setHandle	21
5.2	Helper functions and classes	21
5.2.1	Detailed Description	23
5.2.2	Function Documentation	23
5.2.2.1	ConvertValue	23
5.2.2.2	ConvertValue	23
5.2.2.3	ConvertValue	23
5.2.2.4	ConvertValue	23
5.2.2.5	ConvertValue	24
5.2.2.6	ConvertValue	24

5.2.2.7	ConvertValue	24
5.2.2.8	ConvertValue	24
5.2.2.9	ConvertValue	24
5.2.2.10	ConvertValue	24
5.2.2.11	ConvertValue	25
5.2.2.12	ConvertValue	25
5.2.2.13	emptyMatrix	25
5.2.2.14	pointOnEdge	25
5.2.2.15	pointOnEdge	26
5.2.2.16	RemoveDisallowedCharactersFromName	26
5.3	Input and output	26
5.3.1	Detailed Description	27
5.4	Undo commands	27
5.4.1	Detailed Description	29
5.5	C API	29
5.5.1	Detailed Description	29
5.6	Plotting	29
5.7	TinkerCell plug-ins	30
5.7.1	Detailed Description	30
6	Class Documentation	31
6.1	Tinkercell::AbstractInputWindow Class Reference	31
6.1.1	Detailed Description	32
6.1.2	Constructor & Destructor Documentation	33
6.1.2.1	AbstractInputWindow	33
6.1.3	Member Function Documentation	33
6.1.3.1	exec	33
6.2	Tinkercell::AddControlPointCommand Class Reference	33
6.2.1	Detailed Description	34
6.2.2	Constructor & Destructor Documentation	34
6.2.2.1	AddControlPointCommand	34
6.2.2.2	AddControlPointCommand	35
6.2.3	Member Function Documentation	35
6.2.3.1	redo	35
6.2.3.2	undo	35
6.3	Tinkercell::AddCurveSegmentCommand Class Reference	35
6.3.1	Detailed Description	36

6.3.2	Constructor & Destructor Documentation	37
6.3.2.1	AddCurveSegmentCommand	37
6.3.2.2	AddCurveSegmentCommand	37
6.3.3	Member Function Documentation	37
6.3.3.1	redo	37
6.3.3.2	undo	38
6.4	Tinkercell::ArrowHeadItem Class Reference	38
6.4.1	Detailed Description	39
6.4.2	Constructor & Destructor Documentation	39
6.4.2.1	ArrowHeadItem	39
6.4.2.2	ArrowHeadItem	39
6.4.2.3	ArrowHeadItem	40
6.4.3	Member Function Documentation	40
6.4.3.1	cast	40
6.4.3.2	clone	40
6.4.3.3	paint	40
6.5	Tinkercell::AssignHandleCommand Class Reference	41
6.5.1	Detailed Description	41
6.6	Tinkercell::C_API_Slots Class Reference	41
6.6.1	Detailed Description	42
6.7	Tinkercell::Change2DataCommand< T1, T2 > Class Template Reference	42
6.7.1	Detailed Description	43
6.7.2	Constructor & Destructor Documentation	43
6.7.2.1	Change2DataCommand	43
6.7.2.2	Change2DataCommand	44
6.8	Tinkercell::ChangeBrushAndPenCommand Class Reference	44
6.8.1	Detailed Description	44
6.8.2	Constructor & Destructor Documentation	45
6.8.2.1	ChangeBrushAndPenCommand	45
6.8.2.2	ChangeBrushAndPenCommand	45
6.9	Tinkercell::ChangeBrushCommand Class Reference	45
6.9.1	Detailed Description	46
6.9.2	Constructor & Destructor Documentation	46
6.9.2.1	ChangeBrushCommand	46
6.9.2.2	ChangeBrushCommand	46
6.10	Tinkercell::ChangeDataCommand< T > Class Template Reference	47

6.10.1 Detailed Description	47
6.10.2 Constructor & Destructor Documentation	48
6.10.2.1 ChangeDataCommand	48
6.10.2.2 ChangeDataCommand	48
6.11 Tinkercell::ChangeParentCommand Class Reference	48
6.11.1 Detailed Description	49
6.11.2 Constructor & Destructor Documentation	49
6.11.2.1 ChangeParentCommand	49
6.11.2.2 ChangeParentCommand	49
6.12 Tinkercell::ChangePenCommand Class Reference	50
6.12.1 Detailed Description	50
6.12.2 Constructor & Destructor Documentation	50
6.12.2.1 ChangePenCommand	50
6.12.2.2 ChangePenCommand	50
6.13 Tinkercell::ChangeZCommand Class Reference	51
6.13.1 Detailed Description	51
6.13.2 Constructor & Destructor Documentation	51
6.13.2.1 ChangeZCommand	51
6.13.2.2 ChangeZCommand	52
6.14 Tinkercell::CodeEditor Class Reference	52
6.15 Tinkercell::CommandTextEdit Class Reference	53
6.15.1 Detailed Description	56
6.16 Tinkercell::CompositeCommand Class Reference	56
6.16.1 Detailed Description	57
6.16.2 Constructor & Destructor Documentation	57
6.16.2.1 CompositeCommand	57
6.16.2.2 CompositeCommand	57
6.17 Tinkercell::ConnectionFamily Class Reference	57
6.17.1 Detailed Description	60
6.17.2 Member Function Documentation	60
6.17.2.1 addParticipant	60
6.17.2.2 findValidChildFamilies	60
6.17.2.3 isA	60
6.17.2.4 isValidSet	60
6.17.2.5 numberOfIdenticalNodesFamilies	61
6.17.2.6 participantFamily	61

6.17.2.7	participantRoles	61
6.17.2.8	participantTypes	61
6.17.2.9	synonyms	62
6.18	Tinkercell::ConnectionGraphicsItem Class Reference	62
6.18.1	Detailed Description	67
6.18.2	Constructor & Destructor Documentation	67
6.18.2.1	ConnectionGraphicsItem	67
6.18.2.2	ConnectionGraphicsItem	67
6.18.2.3	ConnectionGraphicsItem	67
6.18.2.4	~ConnectionGraphicsItem	67
6.18.3	Member Function Documentation	67
6.18.3.1	adjustEndPoints	67
6.18.3.2	arrowAt	68
6.18.3.3	arrowHeads	68
6.18.3.4	arrowHeadsAsGraphicsItems	68
6.18.3.5	cast	68
6.18.3.6	cast	69
6.18.3.7	centerLocation	69
6.18.3.8	clear	69
6.18.3.9	clone	69
6.18.3.10	copyPoints	69
6.18.3.11	hideControlPoints	70
6.18.3.12	indexOf	70
6.18.3.13	isModifier	70
6.18.3.14	isValid	70
6.18.3.15	modifierArrowAt	70
6.18.3.16	modifierArrowHeads	71
6.18.3.17	nodeAt	71
6.18.3.18	nodes	71
6.18.3.19	nodesAsGraphicsItems	71
6.18.3.20	nodesDisconnected	72
6.18.3.21	nodesWithArrows	72
6.18.3.22	nodesWithoutArrows	72
6.18.3.23	operator=	72
6.18.3.24	pen	72
6.18.3.25	refresh	73

6.18.3.26	replaceNode	73
6.18.3.27	replaceNodeAt	73
6.18.3.28	setControlPointsVisible	74
6.18.3.29	setPath	74
6.18.3.30	setPen	74
6.18.3.31	shape	74
6.18.3.32	showControlPoints	74
6.18.3.33	slopeAtPoint	75
6.18.3.34	topLevelConnectionItem	75
6.19	Tinkercell::ConnectionHandle Class Reference	75
6.19.1	Detailed Description	77
6.19.2	Constructor & Destructor Documentation	77
6.19.2.1	ConnectionHandle	77
6.19.2.2	ConnectionHandle	77
6.19.3	Member Function Documentation	77
6.19.3.1	addNode	77
6.19.3.2	cast	78
6.19.3.3	cast	78
6.19.3.4	clone	78
6.19.3.5	family	78
6.19.3.6	findValidChildFamilies	79
6.19.3.7	nodes	79
6.19.3.8	nodesIn	79
6.19.3.9	nodesOut	79
6.19.3.10	setFamily	79
6.20	Tinkercell::ConsoleWindow Class Reference	80
6.20.1	Detailed Description	82
6.20.2	Member Function Documentation	82
6.20.2.1	message	82
6.21	Tinkercell::NodeGraphicsItem::ControlPoint Class Reference	82
6.21.1	Detailed Description	83
6.21.2	Member Function Documentation	83
6.21.2.1	clone	83
6.21.2.2	operator=	83
6.21.2.3	paint	84
6.22	Tinkercell::ConnectionGraphicsItem::ControlPoint Class Reference	84

6.22.1	Detailed Description	85
6.22.2	Constructor & Destructor Documentation	85
6.22.2.1	~ControlPoint	85
6.22.3	Member Function Documentation	85
6.22.3.1	clone	85
6.22.3.2	operator=	86
6.23	Tinkercell::ControlPoint Class Reference	86
6.23.1	Detailed Description	88
6.23.2	Member Enumeration Documentation	88
6.23.2.1	"@3	88
6.23.3	Constructor & Destructor Documentation	88
6.23.3.1	ControlPoint	88
6.23.4	Member Function Documentation	88
6.23.4.1	clone	88
6.23.4.2	paint	88
6.23.4.3	rect	88
6.23.4.4	setRect	89
6.24	Tinkercell::Core_FtoS Class Reference	89
6.24.1	Detailed Description	92
6.25	Tinkercell::CThread Class Reference	92
6.25.1	Detailed Description	96
6.25.2	Constructor & Destructor Documentation	96
6.25.2.1	CThread	96
6.25.2.2	CThread	96
6.25.3	Member Function Documentation	96
6.25.3.1	autoUnload	96
6.25.3.2	library	97
6.25.3.3	loadLibrary	97
6.25.3.4	setArg	97
6.25.3.5	setArg	97
6.25.3.6	setArg	97
6.25.3.7	setAutoUnload	98
6.25.3.8	setCharFunction	98
6.25.3.9	setDoubleFunction	98
6.25.3.10	setFunction	98
6.25.3.11	setFunction	98

6.25.3.12	setFunction	98
6.25.3.13	setFunction	99
6.25.3.14	setLibrary	99
6.25.3.15	setLibrary	99
6.25.3.16	setMatrixFunction	99
6.25.3.17	setTitle	99
6.25.3.18	setVoidFunction	99
6.25.3.19	showProgress	100
6.26	Tinkercell::ConnectionGraphicsItem::CurveSegment Class Reference	100
6.26.1	Detailed Description	100
6.27	Tinkercell::DataTable< T > Class Template Reference	100
6.27.1	Detailed Description	104
6.27.2	Member Function Documentation	104
6.27.2.1	at	104
6.27.2.2	at	104
6.27.2.3	at	105
6.27.2.4	at	105
6.27.2.5	columnName	105
6.27.2.6	columnNames	106
6.27.2.7	columns	106
6.27.2.8	hasColumn	106
6.27.2.9	hasRow	106
6.27.2.10	insertColumn	106
6.27.2.11	insertRow	107
6.27.2.12	operator!=	107
6.27.2.13	operator()	107
6.27.2.14	operator()	108
6.27.2.15	operator()	108
6.27.2.16	operator()	108
6.27.2.17	operator()	108
6.27.2.18	operator()	109
6.27.2.19	operator()	109
6.27.2.20	operator()	109
6.27.2.21	operator==	110
6.27.2.22	removeColumn	110
6.27.2.23	removeColumn	110

6.27.2.24	removeRow	110
6.27.2.25	removeRow	111
6.27.2.26	resize	111
6.27.2.27	rowName	111
6.27.2.28	rowNames	112
6.27.2.29	rows	112
6.27.2.30	setColumnName	112
6.27.2.31	setColumnNames	112
6.27.2.32	setRowName	112
6.27.2.33	setRowNames	113
6.27.2.34	swapColumns	113
6.27.2.35	swapColumns	113
6.27.2.36	swapRows	114
6.27.2.37	swapRows	114
6.27.2.38	transpose	114
6.27.2.39	value	114
6.27.2.40	value	115
6.27.2.41	value	115
6.27.2.42	value	116
6.28	Tinkercell::GraphicsScene Class Reference	116
6.28.1	Detailed Description	126
6.28.2	Member Function Documentation	126
6.28.2.1	addItem	126
6.28.2.2	centerOn	126
6.28.2.3	clearSelection	127
6.28.2.4	colorChanged	127
6.28.2.5	contextMenuEvent	127
6.28.2.6	copyItems	128
6.28.2.7	deselect	128
6.28.2.8	deselect	128
6.28.2.9	disableGrid	128
6.28.2.10	enableGrid	129
6.28.2.11	escapeSignal	129
6.28.2.12	filesDropped	129
6.28.2.13	fitAll	129
6.28.2.14	fitInView	129

6.28.2.15	gridSize	130
6.28.2.16	insert	130
6.28.2.17	insert	130
6.28.2.18	itemsAboutToBeInserted	130
6.28.2.19	itemsAboutToBeMoved	131
6.28.2.20	itemsAboutToBeRemoved	131
6.28.2.21	itemsInserted	131
6.28.2.22	itemsMoved	132
6.28.2.23	itemsRemoved	132
6.28.2.24	itemsSelected	132
6.28.2.25	keyPressed	133
6.28.2.26	keyPressEvent	133
6.28.2.27	keyReleased	133
6.28.2.28	keyReleaseEvent	134
6.28.2.29	lastPoint	134
6.28.2.30	lastScreenPoint	134
6.28.2.31	mapToWidget	135
6.28.2.32	mouseDoubleClicked	135
6.28.2.33	mouseDoubleClickEvent	135
6.28.2.34	mouseDragged	136
6.28.2.35	mouseMoved	136
6.28.2.36	mouseMoveEvent	136
6.28.2.37	mouseOnTopOf	137
6.28.2.38	mousePressed	137
6.28.2.39	mousePressEvent	138
6.28.2.40	mouseReleased	138
6.28.2.41	mouseReleaseEvent	138
6.28.2.42	move	139
6.28.2.43	move	139
6.28.2.44	move	139
6.28.2.45	moving	140
6.28.2.46	parentItemChanged	140
6.28.2.47	popIn	140
6.28.2.48	popOut	140
6.28.2.49	populateContextMenu	141
6.28.2.50	print	141

6.28.2.51	remove	141
6.28.2.52	remove	141
6.28.2.53	sceneRightClick	141
6.28.2.54	select	142
6.28.2.55	select	142
6.28.2.56	selected	142
6.28.2.57	selectedRect	143
6.28.2.58	setBrush	143
6.28.2.59	setBrushAndPen	143
6.28.2.60	setBrushAndPen	143
6.28.2.61	setGridSize	143
6.28.2.62	setParentItem	144
6.28.2.63	setParentItem	144
6.28.2.64	setParentItem	144
6.28.2.65	setPen	144
6.28.2.66	setPen	144
6.28.2.67	snapToGrid	144
6.28.2.68	transform	145
6.28.2.69	transform	145
6.28.2.70	visibleRegion	145
6.28.2.71	zoom	145
6.28.2.72	zoomIn	146
6.28.2.73	zoomOut	146
6.28.2.74	ZValue	146
6.29	TinkerCell::GraphicsView Class Reference	147
6.29.1	Detailed Description	148
6.30	TinkerCell::HistoryWindow Class Reference	148
6.30.1	Detailed Description	148
6.31	TinkerCell::InsertGraphicsCommand Class Reference	148
6.31.1	Detailed Description	149
6.31.2	Constructor & Destructor Documentation	149
6.31.2.1	InsertGraphicsCommand	149
6.31.2.2	InsertGraphicsCommand	149
6.32	TinkerCell::InsertHandlesCommand Class Reference	150
6.32.1	Detailed Description	150
6.32.2	Constructor & Destructor Documentation	150

6.32.2.1	InsertHandlesCommand	150
6.32.2.2	InsertHandlesCommand	151
6.33	Tinkercell::ItemData Class Reference	151
6.33.1	Detailed Description	151
6.34	Tinkercell::ItemFamily Class Reference	152
6.34.1	Detailed Description	154
6.34.2	Constructor & Destructor Documentation	154
6.34.2.1	ItemFamily	154
6.34.3	Member Function Documentation	154
6.34.3.1	allChildren	154
6.35	Tinkercell::ItemHandle Class Reference	155
6.35.1	Detailed Description	158
6.35.2	Constructor & Destructor Documentation	158
6.35.2.1	ItemHandle	158
6.35.3	Member Function Documentation	158
6.35.3.1	allChildren	158
6.35.3.2	allGraphicsItems	158
6.35.3.3	depth	159
6.35.3.4	fullName	159
6.35.3.5	hasNumericalData	159
6.35.3.6	hasTextData	159
6.35.3.7	isA	159
6.35.3.8	isA	160
6.35.3.9	isChildOf	160
6.35.3.10	numericalData	160
6.35.3.11	numericalData	160
6.35.3.12	numericalData	161
6.35.3.13	numericalData	161
6.35.3.14	numericalDataNames	161
6.35.3.15	numericalDataTable	161
6.35.3.16	parentOfFamily	162
6.35.3.17	root	162
6.35.3.18	setParent	162
6.35.3.19	textData	162
6.35.3.20	textData	163
6.35.3.21	textData	163

6.35.3.22	textData	163
6.35.3.23	textDataNames	163
6.35.3.24	textDataTable	164
6.36	Tinkercell::LineNumberArea Class Reference	164
6.37	Tinkercell::MainWindow Class Reference	164
6.37.1	Detailed Description	175
6.37.2	Constructor & Destructor Documentation	175
6.37.2.1	MainWindow	175
6.37.2.2	~MainWindow	175
6.37.3	Member Function Documentation	176
6.37.3.1	addTool	176
6.37.3.2	addToolWindow	176
6.37.3.3	addToViewMenu	176
6.37.3.4	allowMultipleViewModes	176
6.37.3.5	changeConsoleBgColor	177
6.37.3.6	changeConsoleErrorMsgColor	177
6.37.3.7	changeConsoleMsgColor	177
6.37.3.8	changeConsoleTextColor	177
6.37.3.9	closeEvent	177
6.37.3.10	colorChanged	178
6.37.3.11	copyItems	178
6.37.3.12	currentNetwork	178
6.37.3.13	currentScene	178
6.37.3.14	currentTextEditor	178
6.37.3.15	currentWindow	179
6.37.3.16	dataChanged	179
6.37.3.17	escapeSignal	179
6.37.3.18	filesLoaded	179
6.37.3.19	functionPointersToMainThread	179
6.37.3.20	getItemsFromFile	180
6.37.3.21	getItemsFromFile	180
6.37.3.22	handleFamilyChanged	180
6.37.3.23	handlesChanged	181
6.37.3.24	historyChanged	181
6.37.3.25	historyStack	181
6.37.3.26	historyWidget	181

6.37.3.27 initializeMenus	182
6.37.3.28 itemsAboutToBeInserted	182
6.37.3.29 itemsAboutToBeMoved	182
6.37.3.30 itemsAboutToBeRemoved	182
6.37.3.31 itemsDropped	183
6.37.3.32 itemsInserted	183
6.37.3.33 itemsInserted	183
6.37.3.34 itemsInsertedSlot	184
6.37.3.35 itemsMoved	184
6.37.3.36 itemsRemoved	184
6.37.3.37 itemsRemoved	184
6.37.3.38 itemsRemovedSlot	185
6.37.3.39 itemsRenamed	185
6.37.3.40 itemsSelected	185
6.37.3.41 keyPressed	186
6.37.3.42 keyReleased	186
6.37.3.43 lineChanged	186
6.37.3.44 loadDefaultPlugins	187
6.37.3.45 loadDynamicLibrary	187
6.37.3.46 loadFiles	187
6.37.3.47 loadNetwork	187
6.37.3.48 mouseDoubleClicked	187
6.37.3.49 mouseDragged	188
6.37.3.50 mouseMoved	188
6.37.3.51 mouseOnTopOf	189
6.37.3.52 mousePressed	189
6.37.3.53 mouseReleased	189
6.37.3.54 networkClosed	190
6.37.3.55 networkClosing	190
6.37.3.56 networkLoaded	190
6.37.3.57 networkOpened	190
6.37.3.58 networks	191
6.37.3.59 networkSaved	191
6.37.3.60 parentHandleChanged	191
6.37.3.61 parentItemChanged	191
6.37.3.62 parse	192

6.37.3.63	prepareNetworkForSaving	192
6.37.3.64	print	192
6.37.3.65	printToFile	192
6.37.3.66	readSettings	192
6.37.3.67	saveNetwork	192
6.37.3.68	saveSettings	193
6.37.3.69	sceneRightClick	193
6.37.3.70	setCursor	193
6.37.3.71	setupFunctionPointers	193
6.37.3.72	setupFunctionPointersSlot	194
6.37.3.73	setupNewThread	194
6.37.3.74	textChanged	194
6.37.3.75	tool	194
6.37.3.76	toolAboutToBeLoaded	195
6.37.3.77	toolLoaded	195
6.37.3.78	tools	195
6.37.3.79	windowChanged	195
6.38	Tinkercell::MergeHandlesCommand Class Reference	196
6.38.1	Detailed Description	196
6.39	Tinkercell::MoveCommand Class Reference	196
6.39.1	Detailed Description	197
6.39.2	Constructor & Destructor Documentation	197
6.39.2.1	MoveCommand	197
6.39.2.2	MoveCommand	198
6.39.2.3	MoveCommand	198
6.39.3	Member Function Documentation	198
6.39.3.1	refreshAllConnectionIn	198
6.40	Tinkercell::MultithreadedSliderWidget Class Reference	198
6.40.1	Detailed Description	201
6.40.2	Constructor & Destructor Documentation	201
6.40.2.1	MultithreadedSliderWidget	201
6.40.2.2	MultithreadedSliderWidget	201
6.40.3	Member Function Documentation	201
6.40.3.1	setSliders	201
6.40.3.2	setVisibleSliders	202
6.40.3.3	setVisibleSliders	202

6.41 Tinkercell::NetworkHandle Class Reference	202
6.41.1 Detailed Description	208
6.41.2 Member Function Documentation	208
6.41.2.1 annotations	208
6.41.2.2 changeData	209
6.41.2.3 changeData	209
6.41.2.4 changeData	209
6.41.2.5 changeData	209
6.41.2.6 changeData	209
6.41.2.7 changeData	209
6.41.2.8 changeData	210
6.41.2.9 changeData	210
6.41.2.10 changeData	210
6.41.2.11 createScene	210
6.41.2.12 createScene	210
6.41.2.13 createTextEditor	211
6.41.2.14 currentScene	211
6.41.2.15 currentTextEditor	211
6.41.2.16 currentWindow	211
6.41.2.17 dataChanged	212
6.41.2.18 editors	212
6.41.2.19 findData	212
6.41.2.20 findData	212
6.41.2.21 findItem	213
6.41.2.22 findItem	213
6.41.2.23 handleFamilyChanged	213
6.41.2.24 handles	213
6.41.2.25 handlesChanged	214
6.41.2.26 historyChanged	214
6.41.2.27 itemsRenamed	214
6.41.2.28 makeUnique	214
6.41.2.29 makeUnique	215
6.41.2.30 makeUnique	215
6.41.2.31 parentHandleChanged	215
6.41.2.32 parseMath	216
6.41.2.33 scenes	216

6.41.2.34	setModelValues	216
6.41.2.35	setModelValues	216
6.41.2.36	setModelValues	217
6.41.2.37	setModelValues	217
6.41.2.38	setWindowTitle	217
6.41.2.39	showScene	217
6.41.2.40	showTextEditor	217
6.41.2.41	updateSymbolsTable	218
6.41.2.42	updateSymbolsTable	218
6.41.2.43	windowTitle	218
6.41.3	Member Data Documentation	218
6.41.3.1	symbolsTable	218
6.42	Tinkercell::NetworkWindow Class Reference	218
6.42.1	Member Function Documentation	220
6.42.1.1	changeEvent	220
6.42.1.2	closeEvent	220
6.42.1.3	focusInEvent	221
6.42.1.4	networkClosed	221
6.42.1.5	networkClosing	221
6.42.1.6	newScene	221
6.42.1.7	newTextEditor	221
6.42.1.8	popIn	222
6.42.1.9	popOut	222
6.42.1.10	resizeEvent	222
6.42.1.11	setAsCurrentWindow	222
6.42.1.12	setFileName	222
6.42.1.13	setWindowTitle	223
6.43	Tinkercell::NodeFamily Class Reference	223
6.43.1	Detailed Description	224
6.43.2	Constructor & Destructor Documentation	224
6.43.2.1	NodeFamily	224
6.43.3	Member Function Documentation	225
6.43.3.1	isA	225
6.44	Tinkercell::NodeGraphicsItem Class Reference	225
6.44.1	Detailed Description	230
6.44.2	Constructor & Destructor Documentation	230

6.44.2.1	NodeGraphicsItem	230
6.44.2.2	NodeGraphicsItem	230
6.44.2.3	NodeGraphicsItem	230
6.44.2.4	~NodeGraphicsItem	231
6.44.3	Member Function Documentation	231
6.44.3.1	cast	231
6.44.3.2	cast	231
6.44.3.3	clear	231
6.44.3.4	clone	231
6.44.3.5	connectedNodes	232
6.44.3.6	connectionsAsGraphicsItems	232
6.44.3.7	connectionsDisconnected	232
6.44.3.8	connectionsWithArrows	232
6.44.3.9	connectionsWithoutArrows	232
6.44.3.10	normalize	232
6.44.3.11	operator=	233
6.44.3.12	polygon	233
6.44.3.13	refresh	233
6.44.3.14	resetBrush	233
6.44.3.15	resetPen	233
6.44.3.16	resetToDefaults	233
6.44.3.17	setAlpha	234
6.44.3.18	shape	234
6.44.3.19	topLevelNodeItem	234
6.45	Tinkercell::NodeHandle Class Reference	234
6.45.1	Detailed Description	235
6.45.2	Constructor & Destructor Documentation	236
6.45.2.1	NodeHandle	236
6.45.2.2	NodeHandle	236
6.45.3	Member Function Documentation	236
6.45.3.1	cast	236
6.45.3.2	cast	236
6.45.3.3	clone	236
6.45.3.4	connections	237
6.45.3.5	family	237
6.45.3.6	setFamily	237

6.46	Tinkercell::PopupListWidgetDelegate Class Reference	237
6.46.1	Detailed Description	238
6.47	Tinkercell::PopupListWidgetDelegateDialog Class Reference	238
6.47.1	Detailed Description	239
6.48	Tinkercell::ProcessThread Class Reference	239
6.48.1	Detailed Description	240
6.48.2	Constructor & Destructor Documentation	240
6.48.2.1	ProcessThread	240
6.48.3	Member Function Documentation	240
6.48.3.1	dialog	240
6.48.3.2	errors	241
6.48.3.3	output	241
6.49	QUndoCommand Class Reference	241
6.50	Tinkercell::RemoveControlPointCommand Class Reference	242
6.50.1	Detailed Description	243
6.50.2	Constructor & Destructor Documentation	243
6.50.2.1	RemoveControlPointCommand	243
6.50.2.2	RemoveControlPointCommand	244
6.50.3	Member Function Documentation	244
6.50.3.1	redo	244
6.50.3.2	undo	244
6.51	Tinkercell::RemoveCurveSegmentCommand Class Reference	244
6.51.1	Detailed Description	245
6.51.2	Constructor & Destructor Documentation	246
6.51.2.1	RemoveCurveSegmentCommand	246
6.51.2.2	RemoveCurveSegmentCommand	246
6.51.3	Member Function Documentation	246
6.51.3.1	redo	246
6.51.3.2	undo	247
6.52	Tinkercell::RemoveGraphicsCommand Class Reference	247
6.52.1	Detailed Description	247
6.52.2	Constructor & Destructor Documentation	248
6.52.2.1	RemoveGraphicsCommand	248
6.52.2.2	RemoveGraphicsCommand	248
6.53	Tinkercell::RemoveHandlesCommand Class Reference	248
6.53.1	Detailed Description	249

6.53.2	Constructor & Destructor Documentation	249
6.53.2.1	RemoveHandlesCommand	249
6.53.2.2	RemoveHandlesCommand	249
6.54	Tinkercell::RenameCommand Class Reference	250
6.54.1	Detailed Description	251
6.54.2	Constructor & Destructor Documentation	251
6.54.2.1	RenameCommand	251
6.54.2.2	RenameCommand	251
6.54.2.3	RenameCommand	252
6.54.2.4	RenameCommand	252
6.54.2.5	RenameCommand	252
6.54.2.6	RenameCommand	253
6.54.2.7	RenameCommand	253
6.54.2.8	RenameCommand	253
6.55	Tinkercell::ReplaceConnectedNodeCommand Class Reference	254
6.55.1	Detailed Description	254
6.55.2	Constructor & Destructor Documentation	254
6.55.2.1	ReplaceConnectedNodeCommand	254
6.56	Tinkercell::ReplaceNodeGraphicsCommand Class Reference	255
6.56.1	Detailed Description	255
6.56.2	Constructor & Destructor Documentation	255
6.56.2.1	ReplaceNodeGraphicsCommand	255
6.56.2.2	ReplaceNodeGraphicsCommand	256
6.57	Tinkercell::ReverseUndoCommand Class Reference	256
6.57.1	Detailed Description	256
6.57.2	Constructor & Destructor Documentation	257
6.57.2.1	ReverseUndoCommand	257
6.58	Tinkercell::SetGraphicsSceneVisibilityCommand Class Reference	257
6.58.1	Detailed Description	258
6.59	Tinkercell::SetHandleFamilyCommand Class Reference	258
6.59.1	Detailed Description	259
6.60	Tinkercell::SetParentHandleCommand Class Reference	259
6.60.1	Detailed Description	260
6.61	Tinkercell::NodeGraphicsItem::Shape Class Reference	260
6.61.1	Detailed Description	261
6.61.2	Constructor & Destructor Documentation	261

6.61.2.1	Shape	261
6.61.2.2	Shape	262
6.61.3	Member Function Documentation	262
6.61.3.1	boundingRect	262
6.61.3.2	operator=	262
6.61.3.3	refresh	262
6.61.3.4	shape	262
6.61.4	Member Data Documentation	263
6.61.4.1	negative	263
6.61.4.2	nodeItem	263
6.62	Tinkercell::SimpleInputWindow Class Reference	263
6.62.1	Detailed Description	265
6.62.2	Constructor & Destructor Documentation	265
6.62.2.1	SimpleInputWindow	265
6.62.2.2	SimpleInputWindow	266
6.62.2.3	SimpleInputWindow	266
6.62.3	Member Function Documentation	266
6.62.3.1	AddOptions	266
6.62.3.2	AddOptions	267
6.62.3.3	CreateWindow	267
6.62.3.4	CreateWindow	267
6.62.3.5	CreateWindow	268
6.62.3.6	exec	268
6.63	Tinkercell::SymbolsTable Class Reference	268
6.63.1	Detailed Description	270
6.63.2	Constructor & Destructor Documentation	270
6.63.2.1	SymbolsTable	270
6.64	Tinkercell::TextEditor Class Reference	270
6.64.1	Detailed Description	274
6.64.2	Member Function Documentation	274
6.64.2.1	insert	274
6.64.2.2	insert	274
6.64.2.3	itemsInserted	274
6.64.2.4	itemsRemoved	274
6.64.2.5	lineChanged	275
6.64.2.6	parse	275

6.64.2.7	popIn	275
6.64.2.8	popOut	275
6.64.2.9	print	275
6.64.2.10	push	275
6.64.2.11	remove	276
6.64.2.12	remove	276
6.64.2.13	setItems	276
6.64.2.14	textChanged	276
6.65	Tinkercell::TextGraphicsItem Class Reference	276
6.65.1	Detailed Description	278
6.65.2	Constructor & Destructor Documentation	278
6.65.2.1	TextGraphicsItem	278
6.65.2.2	TextGraphicsItem	278
6.65.2.3	TextGraphicsItem	279
6.65.2.4	TextGraphicsItem	279
6.65.3	Member Function Documentation	279
6.65.3.1	cast	279
6.65.3.2	closestItem	279
6.65.3.3	setText	279
6.65.3.4	text	280
6.66	Tinkercell::TextUndoCommand Class Reference	280
6.66.1	Detailed Description	280
6.66.2	Constructor & Destructor Documentation	280
6.66.2.1	TextUndoCommand	280
6.67	Tinkercell::Tool Class Reference	281
6.67.1	Detailed Description	283
6.67.2	Constructor & Destructor Documentation	283
6.67.2.1	Tool	283
6.67.3	Member Function Documentation	284
6.67.3.1	currentNetwork	284
6.67.3.2	currentWindow	284
6.67.3.3	getItemsFromFile	284
6.68	Tinkercell::ToolGraphicsItem Class Reference	284
6.68.1	Detailed Description	285
6.68.2	Member Function Documentation	285
6.68.2.1	cast	285

6.69	Tinkercell::TransformCommand Class Reference	286
6.69.1	Detailed Description	286
6.69.2	Constructor & Destructor Documentation	286
6.69.2.1	TransformCommand	286
6.69.2.2	TransformCommand	287
6.70	Tinkercell::Unit Class Reference	287
6.70.1	Detailed Description	287

Chapter 1

TinkerCell Core Library

The TinkerCell Core library is a set of C++ classes that utilize Nokia's Qt Toolkit. The classes provide functions for drawing networks as well as storing information associated with each node and connection in the network. Being built using Qt Toolkit, the Core library makes extensive use of Qt's Signal/Slot framework. When signals are emitted, e.g. `mousePressed(...)`, the signals are received by one or more slots. Slots are functions that respond to the signals. In the Core library, the `MainWindow` class acts like a "signal hub". Numerous Tools classes (aka "plug-ins") implement the slots for processing the `MainWindow`'s signals. The Core library does not do anything by itself, except display the main window. Tools, or plug-ins, perform all the work. The set of plug-ins in the "BasicTools" folder perform numerous tasks such as inserting, highlighting selected items, renaming an item when the text is changed, etc. Other folders such as "ModelingTools" consist of plug-ins that are used to generate dynamic models of biological system. These plug-ins are not part of TinkerCellCore, but they are very important for the TinkerCell application.

The `MainWindow` class provides the top-level window. It is also a "hub" for numerous signals. Any programmer writing a plug-in must be familiar with all of these signals in order to utilize the Core library well. The `MainWindow` holds multiple `NetworkHandle` class instances. The `NetworkHandle` class is basically what defines a "network". The `NetworkHandle` stores a collection of `ItemHandle` instances. The `ItemHandle` class represents individual nodes (`NodeHandle`) or connections (`ConnectionHandle`). It is important to understand that each network can be displayed in multiple windows and each node or connection can be displayed using multiple graphical items on the screen. The `NetworkWindow` class is a single window that represents either the entire network or just part of a network. A `NetworkHandle` contains one or more `NetworkWindow` instances. Each `NetworkWindow` hold either a `GraphicsView` or a `TextEditor`, but never both. Therefore, a "network" (i.e. `NetworkHandle`) can be displayed to the user using one or more graphical diagrams (`GraphicsView`) or text (`TextEditor`).

To understand the design of the Core library, it is imperative to understand `ItemHandle`. To build well-behaved plug-ins, it is imperative to understand how the Core library uses Undo Commands and Signals. It is also important to review the functions available in the `MainWindow`, `GraphicsScene`, and `NetworkHandle` classes.

DataTable<T>

This is a template class that stores a 2 dimensional table, including the row and column headers. The contents of the table can belong to any type. Typically, TinkerCell only uses double and `QString` types because those are the two allowed data types in the `ItemHandle` class. The `DataTable` class is composed of three vectors: the data, the column headers, and the row headers. The class provides functions for obtaining the data values using header names or index values, removing or adding rows and columns, swapping rows and columns, and resizing the table. `NumericalDataTable` is an alias for `DataTable<double>` and `TextDataTable` is for `DataTable<QString>`.

```
NumericalDataTable * dat = new NumericalDataTable;
```

```

dat->resize(10,4);
dat->colName(0) = "column 1";
dat->seRowNames( QStringList() << "row A" << "row B" << "row C" );
dat->value("row A", "column 1") = 10.0;
dat->removeCol(2);
dat->addCol(3,"column 3"); //insert new column at position 3
dat->value("X", "Y") = 5.0; //automatically creates a new row called X
and new column called Y
int r = dat->rows();
int c = dat->cols();
NumericalDataTable dat2 = dat->transpose();

```

Undo Commands

Numerous classes are defined in the [UndoCommands.h](#) file that inherit from [QUndoCommand](#). These classes contain an `undo()` and a `redo()` method. These functions undo and redo a single action without any other side effects. All changes made to a network are generally done using one of these [QUndoCommand](#) classes. Examples of undo command classes include `MoveCommand`, `InsertGraphicsCommand` and `RemoveGraphicsCommand`, `InsertTextCommand` and `RemoveTextCommand`, `ChangeDataCommand`, and `RenameCommand`. There are several more, one for each "atomic" operation. `CompositeCommand` can be used to construct a more complex command from atomic commands. For example, the "paste" operation is a composite command made from `InsertCommand`, `MoveCommand`, and `RenameCommand` (for renaming newly inserted items). Other plug-ins also use the composite command.

The common procedure for using an undo command is as follows:

```

QList<QGraphicsItem*> graphicsItems;
//add some items into graphicsItems
QUndoCommand * cmd = new InsertGraphicsCommand("some informative message
here",graphicsItems,handles);

if (mainWindow && mainWindow->historyStack())
    mainWindow->historyStack()->push(cmd);

```

Alternatively, the `NetworkHandle` class and `GraphicsScene` class provide functions that automatically do the same operations:

```

QList<QGraphicsItem*> graphicsItems;
//add some items into graphicsItems
GraphicsScene * scene = currentScene();
scene->insert("informative message here", graphicsItems);

```

ItemHandle class

This class is arguable the most integral aspect in the TinkerCell Core library. The `ItemHandle` can be thought of as a "package" with four important components: the graphics items for drawing a node or connection, the data table associated with that node or connection, the tools associated with the node of connection, and the family that the node or connection is identified with. The `ItemHandle` is the complete package that is required to obtain all the information about any item in the network. Since TinkerCell networks can be constructed using text of graphics interface, the `ItemHandle` is not required to have graphical items. For networks constructed using the text editor, the data inside each `ItemHandle` is what is most important.

`NodeHandle` and `ConnectionHandle` inherit from `ItemHandle`. For text based models, it is possible to store connections between nodes and connections using `ConnectionHandle::addNode()` method, which takes a `NodeHandle` and an integer describing the "role" of that node in the connection. The interpretation of the "role" is open to the plug-in using it.

Here is a code example, where two graphics items are placed inside a handle, and a new table is added to the handle:

```

NodeHandle * nodeHandle = new NodeHandle;

//make a node item from an XML file
NodeGraphicsItem * node = new NodeGraphicsItem;
NodeGraphicsReader reader;
reader.readXML(node, "mynode.xml");

//make a text graphics item
TextGraphicsItem * text = new TextGraphicsItem("hello world");

//add graphics items to the handle
nodeHandle->graphicsItems << node << text;

nodeHandle->textData("magic word") = "please";
nodeHandle->numericalData("magic numbers", "pi", "value") = 3.141593;
nodeHandle->numericalData("magic numbers", "e", "value") = 2.718282;

//get the entire table
DataTable<qreal> magicNumbers = nodeHandle->numericalDataTable("magic numbers");
//set the entire table
nodeHandle->numericalDataTable("magic numbers") = magicNumbers;

//get list of all tables
nodeHandle->getNumericalDataNames();
nodeHandle->getTextDataNames();

```

ItemHandle contains several functions for conveniently retrieving information or the list of child items. Please see the ItemHandle documentation . Each ItemHandle instance contains a list of pointers to tools, or classes that inherit from class Tool. These tools are associated with this item. When items are selected by a user, the list of contextMenuActions from each of these tools is placed in context menu and the list of graphics items are displayed to the side.

ItemFamily class

The ItemFamily class is used to describe a family that a node or connection belongs in. Nodes and connections are not required to belong in a family. Each family can have multiple parent families. The two main child classes are NodeFamily and ConnectionFamily. NodeFamily stores the default graphics item(s) that is used to draw an item of that family, and ConnectionFamily stores the default arrow head that is used when drawing connections of a given family. The family information is useful for tools in order to distinguish items and insert data tables according to the family of the item.

```

NodeFamily * f1 = new NodeFamily("family A");
NodeFamily * f2 = new NodeFamily("family B");
f2->setParent(f1); //family B is a sub-family of family A

NodeHandle * node = new NodeHandle("x", f2);

if (node->isA("family A")) // will return true
{
}

```

ItemData

The "Data" inside an ItemHandle is an instance of class ItemData. This class is just composed of two hash tables, numericalData and textData. Each hash table maps a string to a DataTable. These hash tables store all the information needed to describe a node or connection. For example, numericalData["parameters"] might contain all the parameters belonging to this item. The data tables inside each item are added by tools, which often use the family information to decide what data tables to insert in a given item. For example, connections might contain textData["rates"] to describe the flux equations whereas nodes of a particular family might contain some other information, such as textData["DNA sequence"]. It is important to note that each entry is a 2D table of strings or numbers; of course, they can be a 1x1 table as well.

MainWindow class

The MainWindow is always the top-most widget that is created in the main() function. The central widget inside the MainWindow is a Tab Widget with windows that can be popped out. Each widget inside the tab widget is a NetworkWindow. Each NetworkWindow can contain a TextEditor or a GraphicsScene. The MainWindow constructor has two arguments for specifying whether the documents should only contain TextEditors or only GraphicsScene or both. Each GraphicsScene is displayed using a GraphicsView.

The MainWindow class inherits from Qt's QMainWindow. The MainWindow has two main functions:

1. Provide the main window for the docking windows, menus, text editors, and drawing canvas
2. Serve as a Signal hub that routes the signals from each scene or text editor to the plug-ins listening to those signals. Thus, the plug-ins do not need to connect to every single scene and text editor; they only need to connect to the MainWindow's signals. These connections are made in a plug-in's setMainWindow() method.

The MainWindow also provides several Slots that are connected to C function pointers via the C_API_Slots class. These functions include find, rename, move, remove, and other functions for changing the data tables within an item in the network.

Nearly all the members in the MainWindow class are public. This includes the three toolbars: 1. toolBarBasic, which stores buttons for basic functions such as new, open, and save; 2. toolBarEdits, which stores buttons such as copy and paste; 3. toolBarForTools, which is intended for other tools. Tools may also add new toolbars using the addToolBar method in QMainWindow. The context menu (mouse right button) for TextEditor and GraphicsScene are also defined in MainWindow. The menus named contextItemsMenu and contextScreenMenu are used by GraphicsScene when items are selected and when no item is selected, resp.. The menus named contextSelectionMenu and contextEditorMenu are used by TextEditor when text is highlighted and when no text is highlighted, resp. Menu items such as file menu, edit menu, settings menu, and view menu are also public, allowing tools to add new actions to them.

When items are inserted or removed from a GraphicsScene or TextEditor, each class emits a signal indicating that graphics item(s) have been removed and text item(s) have been removed, resp. These signals are connected to signals in the MainWindow with the same names. In addition, MainWindow also emits two signals called itemsInserted and itemsRemoved that only contain the ItemHandles instead of the graphics items or text items. Signals that contain only ItemHandles are useful for tools that do not need to know whether the network was constructed using text or graphical interface.

itemsAboutToBeInserted and itemsAboutToBeRemoved: these signals are emitted just before items are inserted or removed from a network, respectively. It can be used to automatically add or remove items from the list. The signal contains a list of QUndoCommands; new commands can be added to this list to perform additional actions along with the insertion event. **itemsInserted and itemsRemoved:** these signals are emitted after items are inserted or removed from a network, respectively. It can be used to modify the items that have been inserted based on the placement of the items or other conditions. It is also used to add tools to the handle::tools list of the new items. **dataChanged:** this signal is emitted whenever any handle's data entry is changed. It is also emitted when items are inserted or removed. This signal can be used to check when a model needs to be updated. Note that undo events are not captured by this signal, which is only captured by historyChanged signal. **historyChanged:** this signal is emitted whenever any recorded change occurs. This signal can be used to check when a model needs to be updated. **networkOpened, networkClosed, and networkChanged:** these signals are emitted whenever a new network is opened, a network has been closed, or a user has clicked on a different network window (respectively). These signals are usually used to reset contents of widgets that display information about a network. **networkOpening and networkClosing:** these signals are sent before opening or closing networks (respectively). They can be used to check if the network has been saved. **mousePressed, mouseReleased, mouseDragged, mouseDoubleClicked, sceneRightClicked:** These signals are emitted due to mouse events. These signals are emitted even if the useDefaultBehavior switch is off in GraphicsScene. **keyPressed, keyReleased:** These signals are emitted due to keyboard events. These signals are emitted even if the useDefaultBehavior switch is off in GraphicsScene.

NetworkHandle

The NetworkHandle is used to store all the information inside a network. The three main components of a NetworkHandle are: historyStack, symbolsTable, and networkWindows. The history stack is used to store the QUndoCommands that provide the undo/redo capabilities. The symbolsTable stores all the nodes and connections in the network. The list networkWindows stores all the windows that are used to display the network to the user. The NetworkHandle provides convenience functions such as changeData(...) or rename(...). These functions create a [QUndoCommand](#), add it to the history stack. Each NetworkHandle can be represented using one or more windows. All of these windows are connected to the same symbols table and the same history stack. NetworkHandle also contains functions such as find() for finding any string in the network and parseMath for validating a mathematical expression (uses muparser).

NetworkWindow

The NetworkWindow is a window (QMainWindow) inside the MainWindow's tab widget. This window can contain either a TextEditor or a GraphicsScene, but not both. Each NetworkWindow can contain its own toolbar or dock widgets. Each NetworkWindow has functions for replacing its current scene or text editor (warning: this operation cannot be undone). Each NetworkWindow can contain an ItemHandle pointer. This handle can be used for multiple purposes. It is designed for particular scenarios in which each individual window is associated with a handle. By default, this pointer is zero.

SymbolsTable

The SymbolsTable class is used to store all the string found in a network model. These strings include the node and connection names and the row names and column names of all the data contained within each node and connection. The purpose of the symbols table is to easily look-up a symbol and find the network objects associated with that symbol. The symbols table keeps a hash table of names and pointers to the node or connection with that name.

The SymbolsTable is also used to get all the ItemHandles in a network, except for "hidden" ItemHandles. ItemHandles represent objects in a network, whether the model is represented as text or graphics.

Full names are always unique, e.g. Cell1.p1. Just the first name, e.g p1, need not be unique. The symbols table keeps a one-to-one hash table that maps full names to object pointers and a one-to-many that maps the first names to object pointers. The uniqueData hash table stores prefixed strings, e.g. p1.param1, as well as non-prefixed strings, e.g. param1. For each string, the hash table stores all the objects that contain that string and the name of the data table which contains that string.

Each NetworkWindow contains one SymbolsTable instance. This instance is updated during any change (history update) to the network.

GraphicsScene

The GraphicsScene class is used to construct a network visually. It is one of the largest classes in Tinker-Cell. The GraphicsScene inherits from Qt's QGraphicsScene. The primary duty of the GraphicsScene class is to receive mouse and keyboard events and emit necessary signals such as itemsSelected, itemsMoved, or mouseOverItem.

The GraphicsScene also handles selection of objects on the scene and moving objects on the scene. The selected objects are placed in the selected() list, and the moving objects are placed in the moving() list. These lists can be modified by plug-ins in order to modify which objects are selected or moved. Moving items are always grouped together when moving; this makes the movement smoother. For example, if a node has other nodes attached to it, a plug-in can ensure that all the nodes move together by adding each node to the moving() list when any one of them is selected. The GraphicsScene's selection and moving operations can be disabled by setting useDefaultBehavior = false.

In addition to emitting signals and handling selection and moving, the GraphicsScene houses numerous functions for conveniently making changes to a network. The functions include insert, remove, move, rename, and changeData. Each of these functions do three things: make a [QUndoCommand](#) object, push the undo command to the history stack, and emit the necessary signal(s) such as itemsInserted or itemsRe-

moved.

The GraphicsScene is always contained inside a NetworkWindow. Therefore it uses the parent NetworkWindow's history stack and symbols table. Many functions such as `changeData`, `rename`, or `allHandles` simply call the parent NetworkWindow's function.

Configuring GraphicsScene

Various visual features, such as the color of the selection rectangle in a scene and default grid size can be set using global variables: `GraphicsScene::SelectionRectangleBrush`, `GraphicsScene::SelectionRectanglePen`, `GraphicsScene::BackgroundBrush`, `GraphicsScene::ForegroundBrush`, `GraphicsScene::GRID`, `GraphicsScene::GridPen`. `GraphicsScene::MIN_DRAG_DISTANCE` can be used to set the minimum distance that is considered a valid drag, i.e. moving the mouse less than this distance will be considered an accidental movement of the mouse and ignored.

GraphicsView

The GraphicsView is a class for viewing a GraphicsScene. It inherits from `QGraphicsView`, and provides a few extra features such as drag-and-drop and zooming.

Graphics items

Qt's `QGraphicsItem` class is used to draw all the items in the GraphicsScene. The two main graphics item classes are `NodeGraphicsItem` and `ConnectionGraphicsItem`. Supporting graphics items are `TextGraphicsItem` and `ControlPoint`.

The `qgraphicsitem_cast<class>` function can be used to cast a generic `QGraphicsItem` to one of these four classes. In addition, `NodeGraphicsItem::cast` and `ConnectionGraphicsItem::cast` can also be used to get the top-most node or connection item from a generic `QGraphicsItem` instance. Each `NodeGraphicsItem` and `ConnectionGraphicsItem` also contains a string named `ClassType`, which is used to statically cast sub-classes of `Node` or `Connection`. For example, `ArrowHeadItem` is a `NodeGraphicsItem` with `classType = "Arrow Head Item"`. example usage: `if (node->className == ArrowHeadItem::CLASSNAME) static_cast<ArrowHeadItem*>(node)`

ControlPoint

The `ControlPoint` class is used to identify key locations of a `NodeGraphicsItem` or `ConnectionGraphicsItem` that can be used to change the appearance of that item. For example, `NodeGraphicsItem` uses control points around its bounding box, allowing a user to drag the control points in order to resize the item. `ConnectionGraphicsItem` uses control points to define the line or beziers used to draw the connection. See image to the right: the small squares and circles are control points. Control points are generally not child items of the item that they belong with. The two main sub-classes of `ControlPoint` are `NodeGraphicsItem::ControlPoint` and `ConnectionGraphicsItem::ControlPoint`.

NodeGraphicsItem

This class is used to draw nodes on the GraphicsScene. `NodeGraphicsItem` inherits from `QGraphicsItemGroup`, which is used to group several graphics items together. Each `NodeGraphicsItem` is a set of points and a set of shapes that are defined using those points. The points belong to the `ControlPoint` class and the shapes belong to the `Shape` class. The entire `NodeGraphicsItem` can be saved as an XML file using `NodeGraphicsItemWriter` (and `NodeGraphicsItemReader` for reading the XML). The XML file uses the SBML render extension format, which is similar to SVG.

The `NodeGraphicsItem` has convenient functions such as `connections()`. The set of connections connected to a given node is retrieved by looking at the control points that are child items of that node. Each connection must have a control point that is the child item of the node that it is connected to.

Shape This class is a polygon constructed using lines, beziers, or arcs. The `Shape` class inherits from `QGraphicsPolygonItem`. The polygon must be closed. The `refresh()` method is used whenever the shape's control points are changed. This updates the shape's polygon.

ConnectionGraphicsItem

This class is used to draw connections between nodes. ConnectionGraphicsItem is composed of a list of CurveSegment instances. Each CurveSegment is a collection of control points that define a single path, usually with the same central control point. Each curve segment also has two arrow head items -- one at either ends (they can be null). If there is a node at the end of any of the paths, then the control points at the end will be child items (see QGraphicsItem) of that node; so, looking at the parent items of each of the control points at the ends is the correct way to find all the nodes that are connected by a connection.

The ConnectionGraphicsItem also contains an optional centerRegionItem, which is a node that sits at the center of the connection. This node is used when one connection item needs to connect to another connection item. Since connections can only be connected to nodes, the center region item is used when connecting a connection to another.

The control points that constitute a connection are generally parent-free, except for the end control points. As mentioned earlier, if a control point is at the end of a connection and is connected to a node, then the control point will be set as the child of the node item. This allows the control point to move along with the node. The ConnectionGraphicsItem class retrieves all the nodes that it is connected to by looking at the parent items of each of its end control points. ConnectionGraphicsItem provides convenient functions such as nodes(), nodesWithArrows(), nodesWithoutArrows(), where "WithArrows" means that there is an arrow head at the arc leading to the node. It is important to understand that these functions do not imply that the curve segments represent a reaction or some other specific process. They indicate the visual representation, which is then translated to more specific meanings by the plug-ins.

refresh() is used whenever the connection is changed. This function updates the arcs and the shape() of the connection using the control point positions.

The ConnectionGraphicsReader and Writer can be used to read and write a connection item to an XML file.

The default arrow head can be set using ConnectionGraphicsItem::DefaultArrowHeadFile. Similarly, the default middle item (the box at the center) can be set using ConnectionGraphicsItem::DefaultMiddleItemFile. For example:

```
ConnectionGraphicsItem::DefaultArrowHeadFile = appDir + QString("/ArrowItems/Reaction.xml");
ConnectionGraphicsItem::DefaultMiddleItemFile = appDir + QString("/OtherItems/simplecircle.xml");
```

TextEditor class

TextEditor

The TextEditor class is used to construct a network using a text-based language. The syntax is not defined by TextEditor and must be provided by a supporting plug-in. The supporting plug-in is expected to make use of the lineChanged(...) and textChanged(...) signals emitted by TextEditor to identify changes by a user and call the insertItem(...), removeItem(...), or setItem(...) methods in order to modify the network.

Tool (plug-in)

Tool is the parent class for all TinkerCell "plug-ins". The most important method in the Tool class is setMainWindow(), which is used by a new tool to connect with the MainWindow's signals and slots.

Each Tool can choose to create instances of Tool::GraphicsItem and place them on the scene. When these graphics items are selected by the user, TinkerCell Core will call the select(int) method of the Tool that is associated with the graphics item.

Console Window

The ConsoleWindow class provides a generic framework for Tools to receive command-line input as well as display messages or execute commands. Each tool can access the ConsoleWindow using console() or mainWindow->console(). For example:

Tools can also interact with the user by connecting to the ConsoleWindow's commandExecuted signal.

This signal is emitted whenever the user pressed <return> after entering a text at the command prompt. The Tools can process the string and carry out necessary operations.

```

    if (console())
    {
        console()->message("hello world");    //print a message on the co
nsole window
        console()->error("incorrect response"); //print an error message
on the console window
        console()->eval("print 1+2"); //evaluate this expression (only r
uns if a plugin such as python plugin is available)
    }

    DataTable<double> data;
    console()->printTable(data); //print a table (tab-delimited)

    ConsoleWindow * console = console();
    if (console)
    {
        connect(editor, SIGNAL( commandExecuted(const QString& ) ),
                this, SLOT( commandExecuted(const QString& ) ));
    }

```

Tools may also disable and re-enable the ConsoleWindow while they are processing the command by using:

```

    console()->freeze();    //lock the console window
    console()->unfreeze(); //unlock the console window

    Alternatively, Tools may also connect with the freeze() and unfreeze() sl
ots:

    CommandTextEdit * editor = console()->editor();
    if (editor)
    {
        connect(this, SIGNAL(freeze()), editor, SLOT(freeze()));
        connect(this, SIGNAL(unfreeze()), editor, SLOT(unfreeze()));
        connect(this, SIGNAL(setFreeze(bool)), editor, SLOT(setFreeze(bool
))) );
        connect(editor, SIGNAL( commandExecuted(const QString& ) ),
                this, SLOT( commandExecuted(const QString& ) ));
    }

```

CThread

This class is used to run C plugins as separate threads.

InterpreterThread

This class inherits from CThread. It is used to run interpreters such as Python and Octave interpreter.

PythonInterpreterThread

This class inherits from InterpreterThread. It is used to embed Python interpreter. This class uses the C program python/runpy.c.in

OctaveInterpreterThread

This class inherits from CThreads. It is used to embed Octave interpreter. This class uses the C++ program octave/runOctave.cpp (for embedding Octave) and assumes that SWIG has been used to generate tinkercell.oct library (which extends Octave).

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

TinkerCell Core classes	17
Helper functions and classes	21
Input and output	26
Undo commands	27
C API	29
Plotting	29
TinkerCell plug-ins	30

Chapter 3

Class Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Tinkercell::C_API_Slots	41
Tinkercell::CodeEditor	52
Tinkercell::TextEditor	270
Tinkercell::CommandTextEdit	53
Tinkercell::ConnectionGraphicsItem	62
Tinkercell::ControlPoint	86
Tinkercell::ConnectionGraphicsItem::ControlPoint	84
Tinkercell::NodeGraphicsItem::ControlPoint	82
Tinkercell::Core_FtoS	89
Tinkercell::CThread	92
Tinkercell::ConnectionGraphicsItem::CurveSegment	100
Tinkercell::DataTable< T >	100
Tinkercell::GraphicsScene	116
Tinkercell::GraphicsView	147
Tinkercell::HistoryWindow	148
Tinkercell::ItemData	151
Tinkercell::ItemFamily	152
Tinkercell::ConnectionFamily	57
Tinkercell::NodeFamily	223
Tinkercell::ItemHandle	155
Tinkercell::ConnectionHandle	75
Tinkercell::NodeHandle	234
Tinkercell::LineNumberArea	164
Tinkercell::MainWindow	164
Tinkercell::MultithreadedSliderWidget	198
Tinkercell::NetworkHandle	202
Tinkercell::NetworkWindow	218
Tinkercell::NodeGraphicsItem	225
Tinkercell::ArrowHeadItem	38
Tinkercell::PopupListWidgetDelegate	237
Tinkercell::PopupListWidgetDelegateDialog	238
Tinkercell::ProcessThread	239

QUndoCommand	241
Tinkercell::AddControlPointCommand	33
Tinkercell::AddCurveSegmentCommand	35
Tinkercell::AssignHandleCommand	41
Tinkercell::Change2DataCommand< T1, T2 >	42
Tinkercell::ChangeBrushAndPenCommand	44
Tinkercell::ChangeBrushCommand	45
Tinkercell::ChangeDataCommand< T >	47
Tinkercell::ChangeParentCommand	48
Tinkercell::ChangePenCommand	50
Tinkercell::ChangeZCommand	51
Tinkercell::CompositeCommand	56
Tinkercell::InsertGraphicsCommand	148
Tinkercell::InsertHandlesCommand	150
Tinkercell::MergeHandlesCommand	196
Tinkercell::MoveCommand	196
Tinkercell::RemoveControlPointCommand	242
Tinkercell::RemoveCurveSegmentCommand	244
Tinkercell::RemoveGraphicsCommand	247
Tinkercell::RemoveHandlesCommand	248
Tinkercell::RenameCommand	250
Tinkercell::ReplaceConnectedNodeCommand	254
Tinkercell::ReplaceNodeGraphicsCommand	255
Tinkercell::ReverseUndoCommand	256
Tinkercell::SetGraphicsSceneVisibilityCommand	257
Tinkercell::SetHandleFamilyCommand	258
Tinkercell::SetParentHandleCommand	259
Tinkercell::TextUndoCommand	280
Tinkercell::TransformCommand	286
Tinkercell::NodeGraphicsItem::Shape	260
Tinkercell::SymbolsTable	268
Tinkercell::TextGraphicsItem	276
Tinkercell::Tool	281
Tinkercell::AbstractInputWindow	31
Tinkercell::SimpleInputWindow	263
Tinkercell::ConsoleWindow	80
Tinkercell::ToolGraphicsItem	284
Tinkercell::Unit	287

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

TinkerCell::AbstractInputWindow (Classes that inherit from this class can be used as GUI windows that provide interface to C programs (library files))	31
TinkerCell::AddControlPointCommand (An command that adds a new control point to a connection item; it has undo and redo functionality)	33
TinkerCell::AddCurveSegmentCommand (An command that adds a new control point to a connection item; it has undo and redo functionality)	35
TinkerCell::ArrowHeadItem (A node graphics item that is used to draw arrow heads on connection items)	38
TinkerCell::AssignHandleCommand (This command assigns handles to items)	41
TinkerCell::C_API_Slots (A set of slots that are called by C libraries)	41
TinkerCell::Change2DataCommand< T1, T2 > (Changes two different data tables)	42
TinkerCell::ChangeBrushAndPenCommand (This command changes the pen and/or brush of an item)	44
TinkerCell::ChangeBrushCommand (This command changes the brush of an item)	45
TinkerCell::ChangeDataCommand< T > (This template class allows undo and redo of a change made to a data table)	47
TinkerCell::ChangeParentCommand (This command changes the parent of a graphics item (not handles))	48
TinkerCell::ChangePenCommand (This command changes the pen of an item)	50
TinkerCell::ChangeZCommand (This command changes the pen of an item)	51
TinkerCell::CodeEditor	52
TinkerCell::CommandTextEdit (A command-line type text box that other tools can use for scripting interface)	53
TinkerCell::CompositeCommand (This command can be used to combine multiple commands into one command)	56
TinkerCell::ConnectionFamily (This class defines the family of a connection. Inherits from ItemFamily It contains a list of ConnectioGraphicsItems that is the default for this family of connections)	57
TinkerCell::ConnectionGraphicsItem (A graphics nodes item that draws connection between two or more nodes and the arrow heads at the ends)	62

Tinkercell::ConnectionHandle (The handles are used to bring together data and graphics items. Connection Handle contains pointers to all the graphics items that belong to it, the tools that apply to this item, the data for this item, the family that it belongs with, and pointers to nodes connected (in and out))	75
Tinkercell::ConsoleWindow (Used to create an output window that can display outputs)	80
Tinkercell::NodeGraphicsItem::ControlPoint (Control point with a pointer to a NodeGraphicsItem)	82
Tinkercell::ConnectionGraphicsItem::ControlPoint (A control point with a pointer to a ConnectionGraphicsItem)	84
Tinkercell::ControlPoint (A simple circle or square that is used for changing specific locations)	86
Tinkercell::Core_FtoS (Function to Signal converter for MainWindow)	89
Tinkercell::CThread (This class is used to run specific functions inside a C dynamic library as a separate thread. The class can be used to load a library or just run a specific function inside an already loaded library. If the library is loaded by this class, the library will be unloaded upon completion on the function. To prevent the automatic unloading, use the setAutoUnload option. Only four types of functions are supported)	92
Tinkercell::ConnectionGraphicsItem::CurveSegment (A set of control points and two arrow heads)	100
Tinkercell::DataTable< T > (DataTable is a 2D vector with row names and column names) . .	100
Tinkercell::GraphicsScene (The primary task of the graphics scene is to draws items. All interactions with the GraphicsScene is done through MainWindow or NetworkHandle . NetworkHandle provides functions such as move, insert, and remove. MainWindow relays all the signals, such as mouse and key events, from the GraphicsScene . So, there is rarely a need to directly interact with the GraphicsScene)	116
Tinkercell::GraphicsView (GraphicsView class that is used to view the contents of a GraphicsScene . The class inherits from QGraphicsView)	147
Tinkercell::HistoryWindow (This is a small class extending QUndoView that manages a stack of undo commands)	148
Tinkercell::InsertGraphicsCommand (This command performs an insert and allows redo/undo of that insert)	148
Tinkercell::InsertHandlesCommand (This command inserts new handles to a NetworkHandle) .	150
Tinkercell::ItemData (This class is used to store information about nodes or connections. It contains a hashtable of data tables, which is used by different tools to store specific data. The versions queue can be used to keep previous versions of the data)	151
Tinkercell::ItemFamily (This class defines the family of a node or connection. The class contains the icon for the family, family name, and minimal data that defines the family. Each family has a name, which is internally converted to an integer (ID) The ID is used to perform isA checks, thus avoiding repeated string matches)	152
Tinkercell::ItemHandle (The ItemHandle represents a complete object in the network, whether it is a node or a connection. The ItemHandle contains the name of the object and pointers to all the QGraphicsItems that are used to represent the object. Tools associated with the object can be stored within the ItemHandle as well. The ItemHandle can also optionally contain an ItemFamily , which can be used to distinguish different types of nodes or connections, if needed. Each ItemHandle can contain one parent. Several functions are available for conveniently getting the parents and children of an ItemHandle)	155
Tinkercell::LineNumberArea	164
Tinkercell::MainWindow (MainWindow is the parent container for all the other widgets in Tinkercell The central widget in MainWindow is a tab widget. Each tab widget can hold a GraphicsView or a TextEditor . One of the main roles of MainWindow is to serve as a signal/slot hub for Tools)	164
Tinkercell::MergeHandlesCommand (This command places all the graphics items inside one handle into the other)	196
Tinkercell::MoveCommand (This command performs a move and allows redo/undo of that move)	196

Tinkercell::MultithreadedSliderWidget (This class is used to run specific functions inside a C dynamic library as a separate thread. Uses CThread to call the C functions)	198
Tinkercell::NetworkHandle (A class that is used to store a network. The network is a collection of Item Handles. The history stack is also a key component of a network. The network can either be represented as text using TextEditor or visualized with graphical items in the GraphicsScene . Each node and connection are contained in a handle, and each handle can either be represented as text or as graphics. The two main components of NetworkWindow are the SymbolsTable and HistoryStack This class provides functions for inserting items, removing items, and changing information inside the model) . . .	202
Tinkercell::NetworkWindow	218
Tinkercell::NodeFamily (This class defines the family of a node. Inherits from ItemFamily . It contains a list of NodeGraphicsItems that is the default for this family of nodes)	223
Tinkercell::NodeGraphicsItem (A simple figure made from one or more polygons. The class can be represented in an XML file)	225
Tinkercell::NodeHandle (The handles are used to bring together data and graphics items. Node Handle contains pointers to all the graphics items that belong to it, the tools that apply to this item, the data for this item, and the family that it belongs with)	234
Tinkercell::PopupListWidgetDelegate (Delegate used inside the SimpleInputWindow)	237
Tinkercell::PopupListWidgetDelegateDialog (Dialog for list widget)	238
Tinkercell::ProcessThread (This class is used to run a process (command + args) as a separate thread as a separate thread)	239
QUndoCommand	241
Tinkercell::RemoveControlPointCommand (A command that removed control points. Allows undo and redo)	242
Tinkercell::RemoveCurveSegmentCommand (A command that removed control points. Allows undo and redo)	244
Tinkercell::RemoveGraphicsCommand (This command performs an removal and allows redo/undo of that removal)	247
Tinkercell::RemoveHandlesCommand (This command inserts new handles to a NetworkHandle)	248
Tinkercell::RenameCommand (This command changes the name of the handle of an item. important: use full name of the items!)	250
Tinkercell::ReplaceConnectedNodeCommand (This command replaces one node item in a connection item with another)	254
Tinkercell::ReplaceNodeGraphicsCommand (This command can be used to replace the graphical representation of a node from an xml file)	255
Tinkercell::ReverseUndoCommand (This command can be used to invert another undo command (i.e. flip the redo/undo))	256
Tinkercell::SetGraphicsSceneVisibilityCommand (This command is used to hide graphics items. Hidden graphics items will be part (unless their handles are also hidden) of the network but not visible on the screen)	257
Tinkercell::SetHandleFamilyCommand (This command is used to hide graphics items. Hidden graphics items will be part (unless their handles are also hidden) of the network but not visible on the screen)	258
Tinkercell::SetParentHandleCommand (This command assigns parent(s) to one or more handles)	259
Tinkercell::NodeGraphicsItem::Shape (A closed polygon path made from arcs, lines, and beziers)	260
Tinkercell::SimpleInputWindow (Used to create an input window that can receive user inputs for C plugins)	263
Tinkercell::SymbolsTable (The symbols table is updated every time the scene or text editor changes. The symbols table contains the list of item names and ItemHandle pointers as well as names and pointers to each data entry in each item)	268
Tinkercell::TextEditor (This is the window that allows used to construct networks using text, as opposed to graphics, which is done by GraphicsScene . The TextEditor requires a supporting tool that parses the text and calls the itemsInserted or itemsRemoved methods. Without a supporting parser tool, the TextEditor will not do anything)	270

Tinkercell::TextGraphicsItem (Editable text item)	276
Tinkercell::TextUndoCommand (This command performs a text change)	280
Tinkercell::Tool (Everything other than the main window is a tool)	281
Tinkercell::ToolGraphicsItem (Tools that are drawn on the scene instead of displayed as a window)	284
Tinkercell::TransformCommand (This command changes the size, angle, and orientation of an item)	286
Tinkercell::Unit (A unit of measurement)	287

Chapter 5

Module Documentation

5.1 TinkerCell Core classes

The main classes in TinkerCell Core. These form the base for all the plug-ins.

Classes

- class [TinkerCell::ArrowHeadItem](#)
A node graphics item that is used to draw arrow heads on connection items.
- class [TinkerCell::ConnectionGraphicsItem](#)
A graphics nodes item that draws connection between two or more nodes and the arrow heads at the ends.
- class [TinkerCell::ConnectionGraphicsItem::ControlPoint](#)
A control point with a pointer to a [ConnectionGraphicsItem](#).
- class [TinkerCell::ConnectionGraphicsItem::CurveSegment](#)
A set of control points and two arrow heads.
- class [TinkerCell::ControlPoint](#)
A simple circle or square that is used for changing specific locations.
- class [TinkerCell::ProcessThread](#)
This class is used to run a process (command + args) as a separate thread as a separate thread.
- class [TinkerCell::DataTable< T >](#)
[DataTable](#) is a 2D vector with row names and column names.
- class [TinkerCell::GraphicsScene](#)
The primary task of the graphics scene is to draws items. All interactions with the [GraphicsScene](#) is done through [MainWindow](#) or [NetworkHandle](#). [NetworkHandle](#) provides functions such as move, insert, and remove. [MainWindow](#) relays all the signals, such as mouse and key events, from the [GraphicsScene](#). So, there is rarely a need to directly interact with the [GraphicsScene](#).
- class [TinkerCell::GraphicsView](#)

GraphicsView class that is used to view the contents of a *GraphicsScene*. The class inherits from *QGraphicsView*.

- class *Tinkercell::Unit*

A unit of measurement.

- class *Tinkercell::ItemFamily*

This class defines the family of a node or connection. The class contains the icon for the family, family name, and minimal data that defines the family. Each family has a name, which is internally converted to an integer (ID) The ID is used to perform isA checks, thus avoiding repeated string matches.

- class *Tinkercell::NodeFamily*

*This class defines the family of a node. Inherits from *ItemFamily*. It contains a list of *NodeGraphicsItems* that is the default for this family of nodes.*

- class *Tinkercell::ConnectionFactory*

*This class defines the family of a connection. Inherits from *ItemFamily* It contains a list of *ConnectioGraphicsItems* that is the default for this family of connections.*

- class *Tinkercell::ItemHandle*

*The *ItemHandle* represents a complete object in the network, whether it is a node or a connection. The *ItemHandle* contains the name of the object and pointers to all the *QGraphicsItems* that are used to represent the object. Tools associated with the object can be stored within the *ItemHandle* as well. The *ItemHandle* can also optionally contain an *ItemFamily*, which can be used to distinguish different types of nodes or connections, if needed. Each *ItemHandle* can contain one parent. Several functions are available for conviniently getting the parents and children of an *ItemHandle*.*

- class *Tinkercell::NodeHandle*

The handles are used to bring together data and graphics items. Node Handle contains pointers to all the graphics items that belong to it, the tools that apply to this item, the data for this item, and the family that it belongs with.

- class *Tinkercell::ConnectionHandle*

The handles are used to bring together data and graphics items. Connection Handle contains pointers to all the graphics items that belong to it, the tools that apply to this item, the data for this item, the family that it belongs with, and pointers to nodes connected (in and out).

- class *Tinkercell::MainWindow*

MainWindow is the parent container for all the other widgets in *TinkerCell* The central widget in *MainWindow* is a tab widget. Each tab widget can hold a *GraphicsView* or a *TextEditor*. One of the main roles of *MainWindow* is to serve as a signal/slot hub for Tools.

- class *Tinkercell::NetworkHandle*

*A class that is used to store a network. The network is a collection of *Item Handles*. The history stack is also a key component of a network. The network can either be represented as text using *TextEditor* or visualized with graphical items in the *GraphicsScene*. Each node and connection are contained in a handle, and each handle can either be represented as text or as graphics. The two main components of *NetworkWindow* are the *SymbolsTable* and *HistoryStack* This class provides functions for inserting items, removing items, and changing information inside the model.*

- class *Tinkercell::NodeGraphicsItem*

A simple figure made from one or more polygons. The class can be represented in an XML file.

- class [TinkerCell::NodeGraphicsItem::ControlPoint](#)
a control point with a pointer to a [NodeGraphicsItem](#)
- class [TinkerCell::NodeGraphicsItem::Shape](#)
A closed polygon path made from arcs, lines, and beziers.
- class [TinkerCell::SymbolsTable](#)
The symbols table is updated every time the scene or text editor changes. The symbols table contains the list of item names and [ItemHandle](#) pointers as well as names and pointers to each data entry in each item.
- class [TinkerCell::TextEditor](#)
This is the window that allows used to construct networks using text, as opposed to graphics, which is done by [GraphicsScene](#). The [TextEditor](#) requires a supporting tool that parses the text and calls the `itemsInserted` or `itemsRemoved` methods. Without a supporting parser tool, the [TextEditor](#) will not do anything.
- class [TinkerCell::TextGraphicsItem](#)
editable text item
- class [TinkerCell::Tool](#)
everything other than the main window is a tool
- class [TinkerCell::ToolGraphicsItem](#)
tools that are drawn on the scene instead of displayed as a window

Typedefs

- typedef `DataTable< QString >` [TinkerCell::TextDataTable](#)
a numerical data table
- typedef `DataTable< qreal >` [TinkerCell::NumericalDataTable](#)
a numerical data table

Functions

- `QGraphicsItem *` [TinkerCell::getGraphicsItem](#) (`QGraphicsItem *`item)
gets the parent of this item that is a node, text, connection, or control point
- `QGraphicsItem *` [TinkerCell::cloneGraphicsItem](#) (`QGraphicsItem *`item)
Clone a graphics item. The item handle will NOT be duplicated.
- `QList< QGraphicsItem * >` [TinkerCell::cloneGraphicsItems](#) (`QList< QGraphicsItem * >` &items, `QList< ItemHandle * >` &newHandles, bool deep=true)
Clone a list of graphics items.
- `ItemHandle *` [TinkerCell::getHandle](#) (`QGraphicsItem *`)
get the handle from a graphics item

- `QList< ItemHandle * > Tinkercell::getHandle` (const `QList< QGraphicsItem * > &`, bool include-Null=true)
get the handles from graphics items
- void `Tinkercell::setHandle` (`QGraphicsItem *`, `ItemHandle *`)
set the handle of a graphics item (use 0 to remove handle)

5.1.1 Detailed Description

The main classes in TinkerCell Core. These form the base for all the plug-ins.

5.1.2 Function Documentation

5.1.2.1 TINKERCELLEXPORT `QGraphicsItem *` `Tinkercell::cloneGraphicsItem` (`QGraphicsItem * item`)

Clone a graphics item. The item handle will NOT be duplicated.

Parameters

`QGraphicsItem` * a pointer to a `QGraphicsItem`

Returns

`QGraphicsItem *` a `QGraphicsItem` that is clone of the argument

5.1.2.2 TINKERCELLEXPORT `QList< QGraphicsItem * >` `Tinkercell::cloneGraphicsItems` (`QList< QGraphicsItem * > & items`, `QList< ItemHandle * > & newHandles`, bool *deep* = *true*)

Clone a list of graphics items.

Parameters

`QList<QGraphicsItem>`* a list of pointers to a `QGraphicsItems`

`QList<ItemHandle>`* return value: returns all the new handles here

bool duplicate the handles as well (default = true).

Returns

`QList<QGraphicsItem*>` a new list of `QGraphicsItems` that are clones of the corresponding argument

5.1.2.3 TINKERCELLEXPORT `QGraphicsItem *` `Tinkercell::getGraphicsItem` (`QGraphicsItem * item`)

gets the parent of this item that is a node, text, connection, or control point

Parameters

`QGraphicsItem` * Qt graphics item

Returns

QGraphicsItem * node, connection, text, or control point

5.1.2.4 TINKERCELLEXPORT ItemHandle * Tinkercell::getHandle (QGraphicsItem *)

get the handle from a graphics item

Parameters

*QGraphicsItem** graphics item

Returns

ItemHandle* item handle (0 if none)

5.1.2.5 TINKERCELLEXPORT QList< ItemHandle * > Tinkercell::getHandle (const QList< QGraphicsItem * > &, bool *includeNull* = *true*)

get the handles from graphics items

Parameters

QList<QGraphicsItem>* graphics item

bool include null handles (default=true)

Returns

QList<ItemHandle*> item handles

5.1.2.6 TINKERCELLEXPORT void Tinkercell::setHandle (QGraphicsItem *, ItemHandle *)

set the handle of a graphics item (use 0 to remove handle)

Parameters

*QGraphicsItem** graphics item

*ItemHandle** handle (use 0 to remove handle)

5.2 Helper functions and classes

Helper classes and functions that are used by the core classes.

Classes

- class [Tinkercell::HistoryWindow](#)

This is a small class extending QUndoView that manages a stack of undo commands.

- class [Tinkercell::ItemData](#)

This class is used to store information about nodes or connections. It contains a hashtable of data tables, which is used by different tools to store specific data. The versions queue can be used to keep previous versions of the data.

Functions

- QPointF [Tinkercell::pointOnEdge](#) (const QRectF &rect0, const QPointF &p1, qreal dist, bool straight)
gets the point on the edge of the rect such that it is in the same line as the center of the rect and the point (arg)
- QPointF [Tinkercell::pointOnEdge](#) (const NodeGraphicsItem &node, const QPointF &pt, qreal dist, bool straight)
gets the point on the edge of the shape such that it is in the same line as the center of the rect and the point (arg)
- tc_matrix [Tinkercell::emptyMatrix](#) ()
construct a tc_matrix with 0 rows and columns
- ItemHandle * [Tinkercell::ConvertValue](#) (long)
convert void to [ItemHandle](#) pointer*
- long [Tinkercell::ConvertValue](#) (ItemHandle *)
*convert [ItemHandle](#) pointer to void **
- QList< ItemHandle * > * [Tinkercell::ConvertValue](#) (tc_items)
convert tc_items to [QList](#) of [ItemHandle](#) pointers
- tc_items [Tinkercell::ConvertValue](#) (const QList< ItemHandle * > &)
convert [QList](#) of [ItemHandle](#) pointers to tc_items
- QString [Tinkercell::ConvertValue](#) (const char *)
convert char to [QString](#)*
- const char * [Tinkercell::ConvertValue](#) (const QString &)
*convert [QString](#) to null-terminated char**
- DataTable< QString > * [Tinkercell::ConvertValue](#) (tc_table)
convert tc_table to [DataTable](#) of [QString](#)
- tc_table [Tinkercell::ConvertValue](#) (const DataTable< QString > &)
convert [DataTable](#) of [QString](#)s to tc_table
- DataTable< qreal > * [Tinkercell::ConvertValue](#) (tc_matrix)
convert matrix to datatable<double> (see [DataTable.h](#) and [TC_structs.h](#))
- tc_matrix [Tinkercell::ConvertValue](#) (const DataTable< qreal > &)
convert Datatable<double> to tc_matrix (see [DataTable.h](#) and [TC_structs.h](#))

- QStringList [Tinkercell::ConvertValue](#) (tc_strings)
convert tc_strings to QStringList
- tc_strings [Tinkercell::ConvertValue](#) (const QStringList &)
convert QStringList to tc_strings
- QString [Tinkercell::RemoveDisallowedCharactersFromName](#) (const QString &)
This function replaces disallowed characters in a name string.

5.2.1 Detailed Description

Helper classes and functions that are used by the core classes.

5.2.2 Function Documentation

5.2.2.1 TINKERCELLEXPOR `ItemHandle *` `Tinkercell::ConvertValue (long)`

convert void* to [ItemHandle](#) pointer

Returns

`ItemHandle*`

5.2.2.2 TINKERCELLEXPOR `long` `Tinkercell::ConvertValue (ItemHandle *)`

convert [ItemHandle](#) pointer to void *

Returns

`void*`

5.2.2.3 TINKERCELLEXPOR `tc_strings` `Tinkercell::ConvertValue (const QStringList &)`

convert QStringList to tc_strings

Returns

`tc_strings`

5.2.2.4 TINKERCELLEXPOR `QStringList` `Tinkercell::ConvertValue (tc_strings)`

convert tc_strings to QStringList

Returns

`QStringList`

5.2.2.5 TINKERCELLEXPOR `tc_matrix` Tinkercell::ConvertValue (`const DataTable< qreal > &`)

convert `Datatable<double>` to `tc_matrix` (see [DataTable.h](#) and `TC_structs.h`)

Returns

`tc_matrix`

5.2.2.6 TINKERCELLEXPOR `QString` Tinkercell::ConvertValue (`const char *`)

convert `char*` to `QString`

Returns

`QString`

5.2.2.7 TINKERCELLEXPOR `tc_table` Tinkercell::ConvertValue (`const DataTable< QString > &`)

convert [DataTable](#) of `QString`s to `tc_table`

Returns

`tc_table`

5.2.2.8 TINKERCELLEXPOR `const char *` Tinkercell::ConvertValue (`const QString &`)

convert `QString` to null-terminated `char*`

Returns

null-terminated `char*`

5.2.2.9 TINKERCELLEXPOR `DataTable< QString > *` Tinkercell::ConvertValue (`tc_table`)

convert `tc_table` to [DataTable](#) of `QString`

Returns

`QStringList`

5.2.2.10 TINKERCELLEXPOR `QList< ItemHandle * > *` Tinkercell::ConvertValue (`tc_items`)

convert `tc_items` to `QList` of [ItemHandle](#) pointers

Returns

`QList<ItemHandle*>`

5.2.2.11 TINKERCELLEXPOR `tc_items` `Tinkercell::ConvertValue (const QList< ItemHandle * > &)`

convert `QList` of [ItemHandle](#) pointers to `tc_items`

Returns

`tc_items`

5.2.2.12 TINKERCELLEXPOR `DataTable< qreal > * Tinkercell::ConvertValue (tc_matrix)`

convert matrix to `datatable<double>` (see [DataTable.h](#) and `TC_structs.h`)

Returns

[DataTable](#) of `qreals`

5.2.2.13 TINKERCELLEXPOR `tc_matrix` `Tinkercell::emptyMatrix ()`

construct a `tc_matrix` with 0 rows and columns

Returns

`tc_matrix`

5.2.2.14 TINKERCELLEXPOR `QPointF` `Tinkercell::pointOnEdge (const QRectF & rect0, const QPointF & p1, qreal dist, bool straight)`

gets the point on the edge of the rect such that it is in the same line as the center of the rect and the point (arg)

Parameters

rectangle

point outside rectangle

Returns

the point on the edge of the rectangle

Parameters

QRectF rectangle

QPointF point outside rectangle

Returns

`QPointF` the point on the edge of the rectangle

5.2.2.15 TINKERCELLEXPORT QPointF Tinkercell::pointOnEdge (const NodeGraphicsItem & *node*, const QPointF & *pt*, qreal *dist*, bool *straight*)

gets the point on the edge of the shape such that it is in the same line as the center of the rect and the point (*arg*)

gets the point on the edge of the shape such that it is in the same line as the center of the shape's bounding rect and the point (*arg*)

Parameters

shape

point outside rectangle

Returns

the point on the edge of the shape

Parameters

QPainterPath the shape

QPointF point outside shape

Returns

QPointF the point on the edge of the shape

5.2.2.16 TINKERCELLEXPORT QString Tinkercell::RemoveDisallowedCharactersFromName (const QString &)

This function replaces disallowed characters in a name string.

Parameters

QString original string

5.3 Input and output

Classes that read/write graphics information and data information from/to files as well as serve as input/output devices for C functions.

Classes

- class [Tinkercell::AbstractInputWindow](#)

Classes that inherit from this class can be used as GUI windows that provide interface to C programs (library files).

- class [Tinkercell::SimpleInputWindow](#)

Used to create an input window that can receive user inputs for C plugins.

- class [Tinkercell::CommandTextEdit](#)

A command-line type text box that other tools can use for scripting interface.

- class [Tinkercell::ConsoleWindow](#)

Used to create an output window that can display outputs.

5.3.1 Detailed Description

Classes that read/write graphics information and data information from/to files as well as serve as input/output devices for C functions.

5.4 Undo commands

A set of classes that allow undo/redo (using Qt Undo framework).

Classes

- class [Tinkercell::ChangeDataCommand< T >](#)
This template class allows undo and redo of a change made to a data table.
- class [Tinkercell::Change2DataCommand< T1, T2 >](#)
Changes two different data tables.
- class [Tinkercell::TextUndoCommand](#)
this command performs a text change
- class [Tinkercell::InsertHandlesCommand](#)
this command inserts new handles to a [NetworkHandle](#)
- class [Tinkercell::RemoveHandlesCommand](#)
this command inserts new handles to a [NetworkHandle](#)
- class [Tinkercell::MoveCommand](#)
this command performs a move and allows redo/undo of that move
- class [Tinkercell::InsertGraphicsCommand](#)
this command performs an insert and allows redo/undo of that insert
- class [Tinkercell::RemoveGraphicsCommand](#)
this command performs an removal and allows redo/undo of that removal
- class [Tinkercell::ChangeBrushCommand](#)
this command changes the brush of an item
- class [Tinkercell::ChangePenCommand](#)
this command changes the pen of an item
- class [Tinkercell::ChangeBrushAndPenCommand](#)

this command changes the pen and/or brush of an item

- class [Tinkercell::ChangeZCommand](#)

this command changes the pen of an item

- class [Tinkercell::TransformCommand](#)

this command changes the size, angle, and orientation of an item

- class [Tinkercell::ChangeParentCommand](#)

this command changes the parent of a graphics item (not handles)

- class [Tinkercell::RenameCommand](#)

this command changes the name of the handle of an item. important: use full name of the items!

- class [Tinkercell::CompositeCommand](#)

this command can be used to combine multiple commands into one command

- class [Tinkercell::ReverseUndoCommand](#)

this command can be used to invert another undo command (i.e. flip the redo/undo)

- class [Tinkercell::ReplaceNodeGraphicsCommand](#)

this command can be used to replace the graphical representation of a node from an xml file

- class [Tinkercell::AssignHandleCommand](#)

this command assigns handles to items

- class [Tinkercell::MergeHandlesCommand](#)

this command places all the graphics items inside one handle into the other

- class [Tinkercell::SetParentHandleCommand](#)

this command assigns parent(s) to one or more handles

- class [Tinkercell::SetGraphicsSceneVisibilityCommand](#)

this command is used to hide graphics items. Hidden graphics items will be part (unless their handles are also hidden) of the network but not visible on the screen.

- class [Tinkercell::SetHandleFamilyCommand](#)

this command is used to hide graphics items. Hidden graphics items will be part (unless their handles are also hidden) of the network but not visible on the screen.

- class [Tinkercell::AddControlPointCommand](#)

An command that adds a new control point to a connection item; it has undo and redo functionality.

- class [Tinkercell::RemoveControlPointCommand](#)

A command that removed control points. Allows undo and redo.

- class [Tinkercell::AddCurveSegmentCommand](#)

An command that adds a new control point to a connection item; it has undo and redo functionality.

- class [Tinkercell::RemoveCurveSegmentCommand](#)

A command that removed control points. Allows undo and redo.

- class [Tinkercell::ReplaceConnectedNodeCommand](#)
this command replaces one node item in a connection item with another

Typedefs

- typedef ChangeDataCommand< QString > [Tinkercell::ChangeTextDataCommand](#)
this command is used to replace text data inside a handle
- typedef ChangeDataCommand< qreal > [Tinkercell::ChangeNumericalDataCommand](#)
this command is used to replace numerical data inside a handle

5.4.1 Detailed Description

A set of classes that allow undo/redo (using Qt Undo framework).

5.5 C API

C functions that are provided by the TinkerCell Core library and Plug-ins (tools).

Classes

- class [Tinkercell::C_API_Slots](#)
A set of slots that are called by C libraries.
- class [Tinkercell::CThread](#)
This class is used to run specific functions inside a C dynamic library as a separate thread. The class can be used to load a library or just run a specific function inside an already loaded library. If the library is loaded by this class, the library will be unloaded upon completion on the function. To prevent the automatic unloading, use the setAutoUnload option. Only four types of functions are supported.

5.5.1 Detailed Description

C functions that are provided by the TinkerCell Core library and Plug-ins (tools).

5.6 Plotting

Classes that provide 2D and 3D plotting capabilities. Other tools can use signals and slots from these classes to generate plots. Also includes clustering capabilities.

Classes that provide 2D and 3D plotting capabilities. Other tools can use signals and slots from these classes to generate plots. Also includes clustering capabilities.

5.7 TinkerCell plug-ins

Plug-ins, which are classes that inherit from Tool class, provide the large majority of the important features in TinkerCell.

Classes

- class [Tinkercell::MultithreadedSliderWidget](#)

This class is used to run specific functions inside a C dynamic library as a separate thread. Uses [CThread](#) to call the C functions.

5.7.1 Detailed Description

Plug-ins, which are classes that inherit from Tool class, provide the large majority of the important features in TinkerCell.

Chapter 6

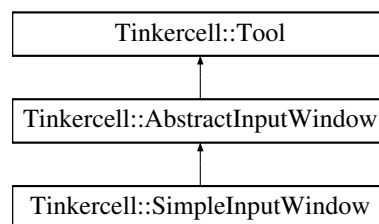
Class Documentation

6.1 Tinkercell::AbstractInputWindow Class Reference

Classes that inherit from this class can be used as GUI windows that provide interface to C programs (library files).

```
#include <AbstractInputWindow.h>
```

Inheritance diagram for Tinkercell::AbstractInputWindow:



Public Slots

- virtual void [escapeSignal](#) (const QWidget *)
Escape signal is a request to stop the current process. This class will hide itself as a response.
- virtual void [exec](#) ()
Executes the [CThread](#).
- virtual void [loadAPI](#) (Tool *)
Uses MainWindow's [setupNewThread](#) function to setup this window's thread.

Signals

- void [updateThread](#) ()
update the thread

- void [evalScript](#) (const QString &)
evaluate a command using command window's eval

Protected Member Functions

- [AbstractInputWindow](#) (const QString &name=tr(""), CThread *thread=0)
constructor
- virtual bool [setMainWindow](#) (MainWindow *main)
Sets the main window. This function will set this tool as a docked widget by default and registered the escapeSignal event. Overwrite this function to prevent that default behavior.
- virtual void [setInput](#) (const DataTable< qreal > &)
set the input for this input window
- virtual void [setThread](#) (CThread *)
set the thread that will be started by this input window
- virtual CThread * [thread](#) () const
the thread that will be started by this input window
- virtual void [enterEvent](#) (QEvent *event)
when mouse enters this widget, the cthread is updated

Protected Attributes

- CThread * [cthread](#)
the target thread
- QDockWidget * [dockWidget](#)
the docked window for this widget (0 if not a docked widget)
- void(* [targetFunction](#))(tc_matrix)
target function for this input window

6.1.1 Detailed Description

Classes that inherit from this class can be used as GUI windows that provide interface to C programs (library files).

See also

LPSolveInput

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Tinkercell::AbstractInputWindow::AbstractInputWindow (const QString & name = `tr("")`, CThread * thread = 0) [protected]

constructor

Parameters

QString name of this tool

CThread the target thread to run from this input window

6.1.3 Member Function Documentation

6.1.3.1 void Tinkercell::AbstractInputWindow::exec () [virtual, slot]

Executes the *CThread*.

See also

CThread

Reimplemented in *Tinkercell::SimpleInputWindow*.

The documentation for this class was generated from the following files:

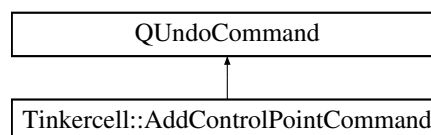
- AbstractInputWindow.h
- AbstractInputWindow.cpp

6.2 Tinkercell::AddControlPointCommand Class Reference

An command that adds a new control point to a connection item; it has undo and redo functionality.

```
#include <UndoCommands.h>
```

Inheritance diagram for Tinkercell::AddControlPointCommand:



Public Member Functions

- *AddControlPointCommand* (const QString &name, *GraphicsScene* *scene, *ConnectionGraphicsItem::ControlPoint* *item)
constructor that makes the command. If added to history stack, also does redo
- *AddControlPointCommand* (const QString &name, *GraphicsScene* *scene, QList< *ConnectionGraphicsItem::ControlPoint* * > items)

constructor that makes the command. If added to history stack, also does redo

- virtual `~AddControlPointCommand()`
destructor: deletes all control points that do not belong a scene
- void `redo()`
Adds a new control point. Control points were set in the constructor.
- void `undo()`
Remove new control points. Control points were set in the constructor.

Public Attributes

- `GraphicsScene * graphicsScene`
graphics scene to which control points were added
- `QList< ConnectionGraphicsItem::ControlPoint * > graphicsItems`
control points that were added
- `QList< int > listK1`
the position(s) at which the control points were added
- `QList< int > listK2`

6.2.1 Detailed Description

An command that adds a new control point to a connection item; it has undo and redo functionality.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 `Tinkercell::AddControlPointCommand::AddControlPointCommand (const QString & name, GraphicsScene * scene, ConnectionGraphicsItem::ControlPoint * item)`

constructor that makes the command. If added to history stack, also does redo

Parameters

name
graphics scene
control point(s) that have been added

Returns

void

6.2.2.2 TinkerCell::AddControlPointCommand::AddControlPointCommand (const QString & name, GraphicsScene * scene, QList< ConnectionGraphicsItem::ControlPoint * > items)

constructor that makes the command. If added to history stack, also does redo

Parameters

name

graphics scene

control point(s) that have been added

Returns

void

6.2.3 Member Function Documentation

6.2.3.1 void TinkerCell::AddControlPointCommand::redo ()

Adds a new control point. Control points were set in the constructor.

Parameters

void

Returns

void

6.2.3.2 void TinkerCell::AddControlPointCommand::undo ()

Remove new control points. Control points were set in the constructor.

Parameters

void

Returns

void

The documentation for this class was generated from the following files:

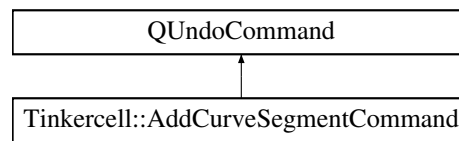
- UndoCommands.h
- UndoCommands.cpp

6.3 TinkerCell::AddCurveSegmentCommand Class Reference

An command that adds a new control point to a connection item; it has undo and redo functionality.

```
#include <UndoCommands.h>
```

Inheritance diagram for TinkerCell::AddCurveSegmentCommand:



Public Member Functions

- [AddCurveSegmentCommand](#) (const QString &name, [GraphicsScene](#) *scene, [ConnectionGraphicsItem](#) *connection, [ConnectionGraphicsItem::CurveSegment](#) &item)
constructor that makes the command. If added to history stack, also does redo
- [AddCurveSegmentCommand](#) (const QString &name, [GraphicsScene](#) *scene, [ConnectionGraphicsItem](#) *connection, QList< [ConnectionGraphicsItem::CurveSegment](#) > items)
constructor that makes the command. If added to history stack, also does redo
- virtual [~AddCurveSegmentCommand](#) ()
destructor. deletes all control points that do not belong a scene
- void [redo](#) ()
Adds a new control point. Control points were set in the constructor.
- void [undo](#) ()
Remove new control points. Control points were set in the constructor.

Public Attributes

- [GraphicsScene](#) * [graphicsScene](#)
graphics scene to which control points were added
- [ConnectionGraphicsItem](#) * [connectionItem](#)
graphics item to which control points were added
- QList< [ConnectionGraphicsItem::CurveSegment](#) > [curveSegments](#)
vector of control points that were added
- QList< int > [listK1](#)
the poission(s) at which the control point vectors were added

6.3.1 Detailed Description

An command that adds a new control point to a connection item; it has undo and redo functionality.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 TinkerCell::AddCurveSegmentCommand::AddCurveSegmentCommand (const QString & *name*, QGraphicsScene * *scene*, ConnectionGraphicsItem * *connection*, ConnectionGraphicsItem::CurveSegment & *item*)

constructor that makes the command. If added to history stack, also does redo

Parameters

name

graphics scene

control point(s) that have been added

Returns

void

6.3.2.2 TinkerCell::AddCurveSegmentCommand::AddCurveSegmentCommand (const QString & *name*, QGraphicsScene * *scene*, ConnectionGraphicsItem * *connection*, QList< ConnectionGraphicsItem::CurveSegment > *items*)

constructor that makes the command. If added to history stack, also does redo

Parameters

name

graphics scene

control point(s) that have been added

Returns

void

6.3.3 Member Function Documentation

6.3.3.1 void TinkerCell::AddCurveSegmentCommand::redo ()

Adds a new control point. Control points were set in the constructor.

Parameters

void

Returns

void

6.3.3.2 void Tinkercell::AddCurveSegmentCommand::undo ()

Remove new control points. Control points were set in the constructor.

Parameters

void

Returns

void

The documentation for this class was generated from the following files:

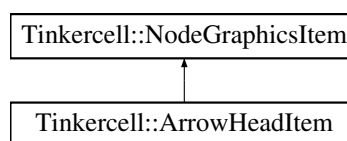
- UndoCommands.h
- UndoCommands.cpp

6.4 Tinkercell::ArrowHeadItem Class Reference

A node graphics item that is used to draw arrow heads on connection items.

```
#include <ConnectionGraphicsItem.h>
```

Inheritance diagram for Tinkercell::ArrowHeadItem:



Public Types

- enum { **Type** = UserType + 6 }
for enabling dynamic_cast

Public Member Functions

- [ArrowHeadItem](#) ([ConnectionGraphicsItem](#) *c=0)
constructor -- initializes the angle and connection item
- [ArrowHeadItem](#) (const QString &, [ConnectionGraphicsItem](#) *c=0)
construct from file
- [ArrowHeadItem](#) (const [ArrowHeadItem](#) &)
copy constructor
- virtual void [paint](#) (QPainter *painter, const QStyleOptionGraphicsItem *option=new QStyleOptionGraphicsItem(), QWidget *widget=0)

paint this arrow item. performs rotation using the angle member.

- virtual [NodeGraphicsItem](#) * [clone](#) () const
returns a duplicate of this arrow head
- virtual int [type](#) () const
for enabling dynamic_cast

Static Public Member Functions

- static [ArrowHeadItem](#) * [cast](#) (QGraphicsItem *)
cast a graphics item to a node graphics item using qgraphicsitem_cast

Public Attributes

- [ConnectionGraphicsItem](#) * [connectionItem](#)
The connection item that this arrow head belongs with.
- qreal [angle](#)
the direction (angle) that the arrow is pointing

Static Public Attributes

- static const QString [CLASSNAME](#) = QString("ArrowHeadItem")
for safe static casting

6.4.1 Detailed Description

A node graphics item that is used to draw arrow heads on connection items.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 TinkerCell::ArrowHeadItem::ArrowHeadItem (ConnectionGraphicsItem * *connection* = 0)

constructor -- initializes the angle and connection item

Constructor: init everything

6.4.2.2 TinkerCell::ArrowHeadItem::ArrowHeadItem (const QString & *filename*, ConnectionGraphicsItem * *connection* = 0)

construct from file

Constructor: init everything

6.4.2.3 Tinkercell::ArrowHeadItem::ArrowHeadItem (const ArrowHeadItem & *copy*)

copy constructor

Constructor: init everything

6.4.3 Member Function Documentation

6.4.3.1 ArrowHeadItem * Tinkercell::ArrowHeadItem::cast (QGraphicsItem * *q*) [static]

cast a graphics item to a node graphics item using qgraphicsitem_cast

Parameters

*QGraphicsItem** graphics item

Returns

ArrowHeadItem* can be 0 if the cast is invalid

Reimplemented from [Tinkercell::NodeGraphicsItem](#).

6.4.3.2 NodeGraphicsItem * Tinkercell::ArrowHeadItem::clone () const [virtual]

returns a duplicate of this arrow head

make a copy of this item

Returns

duplicate arrow head item

Reimplemented from [Tinkercell::NodeGraphicsItem](#).

6.4.3.3 void Tinkercell::ArrowHeadItem::paint (QPainter * *painter*, const QStyleOptionGraphicsItem * *option* = new QStyleOptionGraphicsItem(), QWidget * *widget* = 0) [virtual]

paint this arrow item. performs rotation using the angle member.

Returns

void

Reimplemented from [Tinkercell::NodeGraphicsItem](#).

The documentation for this class was generated from the following files:

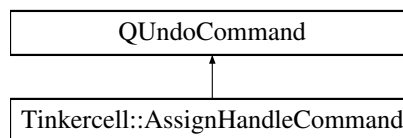
- ConnectionGraphicsItem.h
- ConnectionGraphicsItem.cpp

6.5 TinkerCell::AssignHandleCommand Class Reference

this command assigns handles to items

```
#include <UndoCommands.h>
```

Inheritance diagram for TinkerCell::AssignHandleCommand:



Public Member Functions

- **AssignHandleCommand** (const QString &text, QGraphicsItem *item, [ItemHandle](#) *handle)
- **AssignHandleCommand** (const QString &text, const QList< QGraphicsItem * > &items, [ItemHandle](#) *handle)
- **AssignHandleCommand** (const QString &text, const QList< QGraphicsItem * > &items, QList< [ItemHandle](#) * > &handles)
- void **redo** ()
- void **undo** ()

Public Attributes

- QList< QGraphicsItem * > **graphicsItems**
- QList< [ItemHandle](#) * > **oldHandles**
- QList< [ItemHandle](#) * > **newHandles**

6.5.1 Detailed Description

this command assigns handles to items

The documentation for this class was generated from the following files:

- UndoCommands.h
- UndoCommands.cpp

6.6 TinkerCell::C_API_Slots Class Reference

A set of slots that are called by C libraries.

```
#include <C_API_Slots.h>
```

Signals

- void **saveNetwork** (const QString &)

Public Member Functions

- `C_API_Slots` ([MainWindow](#) *)

6.6.1 Detailed Description

A set of slots that are called by C libraries.

The documentation for this class was generated from the following files:

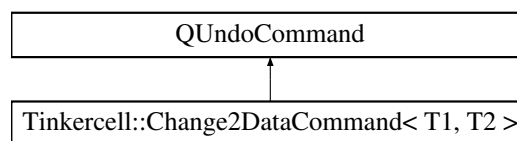
- `C_API_Slots.h`
- `C_API_Slots.cpp`

6.7 Tinkercell::Change2DataCommand< T1, T2 > Class Template Reference

Changes two different data tables.

```
#include <DataTable.h>
```

Inheritance diagram for `Tinkercell::Change2DataCommand< T1, T2 >`:



Public Member Functions

- `Change2DataCommand` (const `QString` &name, [DataTable](#)< `T1` > *oldDataTable1, const [DataTable](#)< `T1` > *newDataTable1, [DataTable](#)< `T2` > *oldDataTable2, const [DataTable](#)< `T2` > *newDataTable2)

constructor

- `Change2DataCommand` (const `QString` &name, const `QList`< [DataTable](#)< `T1` > * > &oldDataTable1, const `QList`< [DataTable](#)< `T1` > * > &newDataTable1, const `QList`< [DataTable](#)< `T2` > * > &oldDataTable2, const `QList`< [DataTable](#)< `T2` > * > &newDataTable2)

constructor

- void `redo` ()

redo the changes

- void `undo` ()

undo the changes

Public Attributes

- `QList< DataTable< T1 > * > targetDataTable1`
target tables of type T1
- `QList< DataTable< T1 > > newDataTable1`
new tables of type T1
- `QList< DataTable< T1 > > oldDataTable1`
old tables of type T1
- `QList< DataTable< T2 > * > targetDataTable2`
target tables of type T2
- `QList< DataTable< T2 > > newDataTable2`
new tables of type T2
- `QList< DataTable< T2 > > oldDataTable2`
old tables of type T2

6.7.1 Detailed Description

`template<typename T1, typename T2> class TinkerCell::Change2DataCommand< T1, T2 >`

Changes two different data tables.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 `template<typename T1, typename T2> TinkerCell::Change2DataCommand< T1, T2 >::Change2DataCommand (const QString & name, DataTable< T1 > * oldDataTable1, const DataTable< T1 > * newDataTable1, DataTable< T2 > * oldDataTable2, const DataTable< T2 > * newDataTable2)`

constructor

Parameters

name of the command

old table of type T1

new table of type T1

old table of type T2

new table of type T2

6.7.2.2 `template<typename T1, typename T2> Tinkercell::Change2DataCommand< T1, T2 >::Change2DataCommand (const QString & name, const QList< DataTable< T1 > * > & oldDataTable1, const QList< DataTable< T1 > * > & newDataTable1, const QList< DataTable< T2 > * > & oldDataTable2, const QList< DataTable< T2 > * > & newDataTable2)`

constructor

Parameters

name of the command

old tables of type T1

new tables of type T1

old tables of type T2

new tables of type T2

The documentation for this class was generated from the following file:

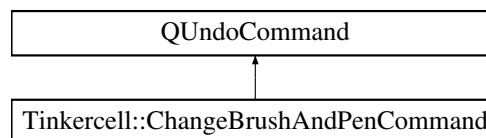
- DataTable.h

6.8 Tinkercell::ChangeBrushAndPenCommand Class Reference

this command changes the pen and/or brush of an item

```
#include <UndoCommands.h>
```

Inheritance diagram for Tinkercell::ChangeBrushAndPenCommand:



Public Member Functions

- [ChangeBrushAndPenCommand](#) (const QString &name, QGraphicsItem *item, const QBrush &brush, const QPen &pen)

constructor

- [ChangeBrushAndPenCommand](#) (const QString &name, const QList< QGraphicsItem * > &items, const QList< QBrush > &brushes, const QList< QPen > &pens)

constructor

- void **redo** ()
- void **undo** ()

6.8.1 Detailed Description

this command changes the pen and/or brush of an item

6.8.2 Constructor & Destructor Documentation

6.8.2.1 TinkerCell::ChangeBrushAndPenCommand::ChangeBrushAndPenCommand (const QString & name, QGraphicsItem * item, const QBrush & brush, const QPen & pen)

constructor

Parameters

QString name of command
*GraphicsScene** scene where change happened
*QGraphicsItem** item that is affected
QBrush new brushes (one for each item)
QPen new pens (one for each item)

6.8.2.2 TinkerCell::ChangeBrushAndPenCommand::ChangeBrushAndPenCommand (const QString & name, const QList< QGraphicsItem * > & items, const QList< QBrush > & brushes, const QList< QPen > & pens)

constructor

Parameters

QString name of command
*GraphicsScene** scene where change happened
QList<QGraphicsItem>&* items that are affected
QList<QBrush>& new brushes (one for each item)
QList<QPen>& new pens (one for each item)

The documentation for this class was generated from the following files:

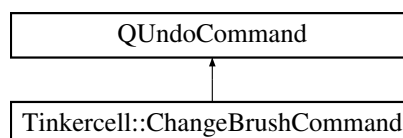
- UndoCommands.h
- UndoCommands.cpp

6.9 TinkerCell::ChangeBrushCommand Class Reference

this command changes the brush of an item

```
#include <UndoCommands.h>
```

Inheritance diagram for TinkerCell::ChangeBrushCommand:



Public Member Functions

- [ChangeBrushCommand](#) (const QString &name, QGraphicsItem *item, const QBrush &to)
constructor
- [ChangeBrushCommand](#) (const QString &name, const QList< QGraphicsItem * > &items, const QList< QBrush > &to)
constructor
- void **redo** ()
- void **undo** ()

6.9.1 Detailed Description

this command changes the brush of an item

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Tinkercell::ChangeBrushCommand::ChangeBrushCommand (const QString & name, QGraphicsItem * item, const QBrush & to)

constructor

Parameters

QString name of command
*GraphicsScene** scene where change happened
*QGraphicsItem** item that is affected
QBrush new brush

6.9.2.2 Tinkercell::ChangeBrushCommand::ChangeBrushCommand (const QString & name, const QList< QGraphicsItem * > & items, const QList< QBrush > & to)

constructor

Parameters

QString name of command
*GraphicsScene** scene where change happened
QList<QGraphicsItem>&* items that are affected
QList<QBrush>& new brushes (one for each item)

The documentation for this class was generated from the following files:

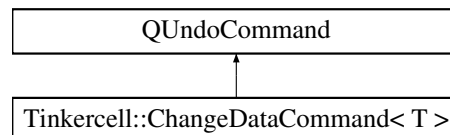
- UndoCommands.h
- UndoCommands.cpp

6.10 TinkerCell::ChangeDataCommand< T > Class Template Reference

This template class allows undo and redo of a change made to a data table.

```
#include <DataTable.h>
```

Inheritance diagram for TinkerCell::ChangeDataCommand< T >:



Public Member Functions

- [ChangeDataCommand](#) (const QString &name, [DataTable](#)< T > *oldDataTable, const [DataTable](#)< T > *newDataTable)
constructor
- [ChangeDataCommand](#) (const QString &name, const QList< [DataTable](#)< T > * > &oldDataTable, const QList< [DataTable](#)< T > * > &newDataTable)
constructor
- void [redo](#) ()
redo the changes
- void [undo](#) ()
undo the changes

Public Attributes

- QList< [DataTable](#)< T > * > [targetDataTable](#)
pointers to target tables
- QList< [DataTable](#)< T > > [newDataTable](#)
new tables
- QList< [DataTable](#)< T > > [oldDataTable](#)
old tables

6.10.1 Detailed Description

```
template<typename T> class TinkerCell::ChangeDataCommand< T >
```

This template class allows undo and redo of a change made to a data table.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 `template<typename T > Tinkercell::ChangeDataCommand< T
>::ChangeDataCommand (const QString & name, DataTable< T > * oldDataTable,
const DataTable< T > * newDataTable)`

constructor

Parameters

name of the change

old tables

new tables

6.10.2.2 `template<typename T > Tinkercell::ChangeDataCommand< T
>::ChangeDataCommand (const QString & name, const QList< DataTable< T > * >
& oldDataTable, const QList< DataTable< T > * > & newDataTable)`

constructor

Parameters

name of the change

old table

new table

The documentation for this class was generated from the following file:

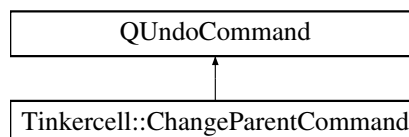
- DataTable.h

6.11 Tinkercell::ChangeParentCommand Class Reference

this command changes the parent of a graphics item (not handles)

```
#include <UndoCommands.h>
```

Inheritance diagram for Tinkercell::ChangeParentCommand:



Public Member Functions

- [ChangeParentCommand](#) (const QString &name, QGraphicsScene *scene, QGraphicsItem *item, QGraphicsItem *newParent)

constructor

- [ChangeParentCommand](#) (const QString &name, QGraphicsScene *scene, const QList< QGraphicsItem * > &items, const QList< QGraphicsItem * > &newParents)

constructor

- void **redo** ()
- void **undo** ()

6.11.1 Detailed Description

this command changes the parent of a graphics item (not handles)

6.11.2 Constructor & Destructor Documentation

6.11.2.1 TinkerCell::ChangeParentCommand::ChangeParentCommand (const QString & name, QGraphicsScene * scene, QGraphicsItem * item, QGraphicsItem * newParent)

constructor

Parameters

QString name of command

*GraphicsScene** scene where change happened

*QGraphicsItem** item that is affected

*QGraphicsItem** new parent item

6.11.2.2 TinkerCell::ChangeParentCommand::ChangeParentCommand (const QString & name, QGraphicsScene * scene, const QList< QGraphicsItem * > & items, const QList< QGraphicsItem * > & newParents)

constructor

Parameters

QString name of command

*GraphicsScene** scene where change happened

*QList<QGraphicsItem * >&* items that are affected

*QList<QGraphicsItem * >&* new parent items

The documentation for this class was generated from the following files:

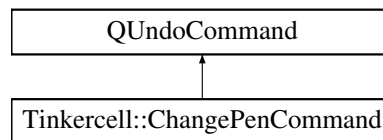
- UndoCommands.h
- UndoCommands.cpp

6.12 TinkerCell::ChangePenCommand Class Reference

this command changes the pen of an item

```
#include <UndoCommands.h>
```

Inheritance diagram for TinkerCell::ChangePenCommand:



Public Member Functions

- [ChangePenCommand](#) (const QString &name, QGraphicsItem *item, const QPen &to)
constructor
- [ChangePenCommand](#) (const QString &name, const QList< QGraphicsItem * > &items, const QList< QPen > &to)
constructor
- void **redo** ()
- void **undo** ()

6.12.1 Detailed Description

this command changes the pen of an item

6.12.2 Constructor & Destructor Documentation

6.12.2.1 TinkerCell::ChangePenCommand::ChangePenCommand (const QString & name, QGraphicsItem * item, const QPen & to)

constructor

Parameters

QString name of command
*GraphicsScene** scene where change happened
*QGraphicsItem** item that is affected
QBrush new pen

6.12.2.2 TinkerCell::ChangePenCommand::ChangePenCommand (const QString & name, const QList< QGraphicsItem * > & items, const QList< QPen > & to)

constructor

Parameters

- QString* name of command
- GraphicsScene** scene where change happened
- QList<QGraphicsItem*>&* items that are affected
- QList<QPen>&* new pens (one for each item)

The documentation for this class was generated from the following files:

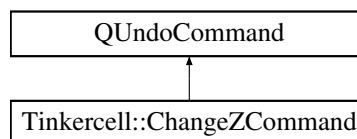
- UndoCommands.h
- UndoCommands.cpp

6.13 TinkerCell::ChangeZCommand Class Reference

this command changes the pen of an item

```
#include <UndoCommands.h>
```

Inheritance diagram for TinkerCell::ChangeZCommand:

**Public Member Functions**

- [ChangeZCommand](#) (const QString &name, QGraphicsScene *scene, QGraphicsItem *item, qreal to)
constructor
- [ChangeZCommand](#) (const QString &name, QGraphicsScene *scene, const QList< QGraphicsItem * > &items, const QList< qreal > &to)
constructor
- void **redo** ()
- void **undo** ()

6.13.1 Detailed Description

this command changes the pen of an item

6.13.2 Constructor & Destructor Documentation

6.13.2.1 TinkerCell::ChangeZCommand::ChangeZCommand (const QString & name, QGraphicsScene * scene, QGraphicsItem * item, qreal to)

constructor

Parameters

QString name of command
*GraphicsScene** scene where change happened
*QGraphicsItem** item that is affected
double new Z value

6.13.2.2 TinkerCell::ChangeZCommand::ChangeZCommand (const QString & name, QGraphicsScene * scene, const QList< QGraphicsItem * > & items, const QList< qreal > & to)

constructor

Parameters

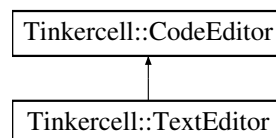
QString name of command
*GraphicsScene** scene where change happened
QList<QGraphicsItem>&* item that is affected
QList<qreal>& new Z (one for each item)

The documentation for this class was generated from the following files:

- UndoCommands.h
- UndoCommands.cpp

6.14 TinkerCell::CodeEditor Class Reference

Inheritance diagram for TinkerCell::CodeEditor:

**Public Slots**

- void **setText** (const QString &)
- void **showFindReplaceDialog** ()
- bool **find** (const QString &)
- bool **replace** (const QString &, const QString &)

Public Member Functions

- **CodeEditor** (QWidget *parent=0)
- void **lineNumberAreaPaintEvent** (QPaintEvent *event)
- int **lineNumberAreaWidth** ()

- void **setCompleter** (QCompleter *c)
- QCompleter * **completer** () const
- void **zoomIn** (int r=1)
- void **zoomOut** (int r=1)
- QString **text** () const

Public Attributes

- QWidget * **lineNumberArea**
- QColor **lineHighlightColor**
- QColor **lineNumberBackground**
- QColor **lineNumberText**

Protected Member Functions

- void **resizeEvent** (QResizeEvent *event)
- virtual void **wheelEvent** (QWheelEvent *wheelEvent)
- void **keyPressEvent** (QKeyEvent *e)
- void **focusInEvent** (QFocusEvent *e)

The documentation for this class was generated from the following files:

- CodeEditor.h
- CodeEditor.cpp

6.15 Tinkercell::CommandTextEdit Class Reference

A command-line type text box that other tools can use for scripting interface.

```
#include <ConsoleWindow.h>
```

Public Slots

- virtual void **eval** (const QString &)
evaluate a command (just emits a commandExecuted signal)
- virtual void **error** (const QString &)
post an error message to this console text box
- virtual void **message** (const QString &)
post a message to this console text box
- virtual void **clearText** ()
clear all text
- virtual void **freeze** ()
equivalent to setFreeze(true)

- virtual void [unfreeze](#) ()
equivalent to `setFreeze(false)`
- virtual void [setFreeze](#) (bool [frozen](#)=true)
Set frozen state. The text box will not respond to user inputs while it is frozen.
- virtual void [setBackground](#)Color (const QColor &)
set background color
- virtual void [setPlainText](#)Color (const QColor &)
set plain text color
- virtual void [setOutputText](#)Color (const QColor &)
set output message color
- virtual void [setErrorText](#)Color (const QColor &)
set error message color
- virtual void [setTableText](#)Color (const QColor &)
set table headers color

Signals

- void [commandExecuted](#) (const QString &command)
the user requested to execute the given command
- void [commandInterrupted](#) ()
the user requested to interrupt the current process

Public Member Functions

- [CommandTextEdit](#) (MainWindow *parent=0)
default constructor
- virtual bool [isFrozen](#) ()
Whether or not this console in the frozen state. The text box will not add or remove text while it is frozen.
- void [setCompleter](#) (QCompleter *c)
set code completion
- QCompleter * [completer](#) () const
code completion

Protected Member Functions

- virtual void [keyPressEvent](#) (QKeyEvent *event)
manages the console-type interface, where the user is not allowed to type outside the >>
- virtual void [wheelEvent](#) (QWheelEvent *wheelEvent)
zoom in or out using mouse wheel
- virtual void [focusInEvent](#) (QFocusEvent *e)
focus returned from code completer

Protected Attributes

- QStringList [historyStack](#)
list of previously executed commands
- QStringList [messagesStack](#)
list of messages pending
- QStringList [errorsStack](#)
list of errors pending
- int [currentHistoryIndex](#)
current position in the history of commands
- int [currentPosition](#)
current position of the cursor in the text box
- bool [frozen](#)
frozen state = 0 or 1
- QTextCharFormat [errorFormat](#)
font format for error messages
- QTextCharFormat [messageFormat](#)
font format for regular messages
- QTextCharFormat [tableHeaderFormat](#)
font format for table headers
- QTextCharFormat [normalFormat](#)
font format for user inputs

6.15.1 Detailed Description

A command-line type text box that other tools can use for scripting interface.

The documentation for this class was generated from the following files:

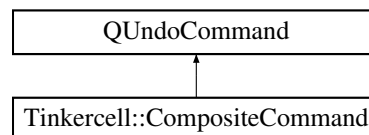
- ConsoleWindow.h
- ConsoleWindow.cpp

6.16 Tinkercell::CompositeCommand Class Reference

this command can be used to combine multiple commands into one command

```
#include <UndoCommands.h>
```

Inheritance diagram for Tinkercell::CompositeCommand:



Public Member Functions

- [CompositeCommand](#) (const QString &, const QList< [QUndoCommand](#) * > &, const QList< [QUndoCommand](#) * > &noClear=QList< [QUndoCommand](#) * >())
Constructor: Composite command takes ownership of these commands unless specified otherwise.
- [CompositeCommand](#) (const QString &, [QUndoCommand](#) *, [QUndoCommand](#) *, bool deleteCommands=true)
constructor for grouping two commands. Composite command takes ownership of these commands unless specified otherwise.
- [~CompositeCommand](#) ()
destructor automatically deletes any command not in the doNotDelete list
- void [redo](#) ()
undo
- void [undo](#) ()
undo

Public Attributes

- QList< [QUndoCommand](#) * > [commands](#)
commands grouped inside this composite command
- QList< [QUndoCommand](#) * > [doNotDelete](#)

commands that should not be deleted along with the composite command

6.16.1 Detailed Description

this command can be used to combine multiple commands into one command

6.16.2 Constructor & Destructor Documentation

6.16.2.1 Tinkercell::CompositeCommand::CompositeCommand (const QString & *name*, const QList< QUndoCommand * > & *list*, const QList< QUndoCommand * > & *noClear* = *QList<QUndoCommand*>()*)

Constructor. Composite command takes ownership of these commands unless specified otherwise.

Parameters

QString name of command

QList<QUndoCommand>&* the commands that make up this composite command

QList<QUndoCommand>&* the commands that should not be deleted by composite command's destructor (default = none)

6.16.2.2 Tinkercell::CompositeCommand::CompositeCommand (const QString & *name*, QUndoCommand * *cmd1*, QUndoCommand * *cmd2*, bool *deleteCommands* = *true*)

constructor for grouping two commands. Composite command takes ownership of these commands unless specified otherwise.

Parameters

QString name of command

*QUndoCommand** a command to be grouped

*QUndoCommand** another command to be grouped

bool delete both commands automatically (default = true)

The documentation for this class was generated from the following files:

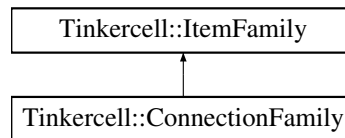
- UndoCommands.h
- UndoCommands.cpp

6.17 Tinkercell::ConnectionFamily Class Reference

This class defines the family of a connection. Inherits from [ItemFamily](#) It contains a list of ConnectionGraphicsItems that is the default for this family of connections.

```
#include <ItemFamily.h>
```

Inheritance diagram for Tinkercell::ConnectionFamily:



Public Member Functions

- virtual [ItemFamily](#) * [parent](#) () const
get the parent for this family. If there are more than one parents, returns the first
- virtual QList< [ItemFamily](#) * > [parents](#) () const
get all the parents for this family.
- virtual QList< [ItemFamily](#) * > [children](#) () const
get all the families that make up this family.
- virtual void [setParent](#) ([ConnectionFactory](#) *)
set parent family
- virtual ~[ConnectionFactory](#) ()
destructor.
- [ConnectionFactory](#) (const QString &name=QString())
constructor.
- virtual bool [isA](#) (const QString &) const
indicates whether or not the given string is the name of this family or any of its parent families
- virtual bool [isA](#) (const [ItemFamily](#) *) const
indicates whether or not the given family is the name of this family or any of its parent families
- virtual bool [addParticipant](#) (const QString &role, const QString &family)
add a participant
- virtual QString [participantFamily](#) (const QString &role) const
get participant family
- virtual QStringList [participantRoles](#) () const
get all participant roles
- virtual QStringList [participantTypes](#) () const
get all participant family names
- virtual bool [isValidSet](#) (const QList< [NodeHandle](#) * > &nodes, bool checkFull=true)
checks if this family is compatible with a connection composed of the given set of nodes
- virtual QList< [ItemFamily](#) * > [findValidChildFamilies](#) (const QList< [NodeHandle](#) * > &, bool checkFull=true)
find child-families of this family that the given set of nodes can potentially belong with

- virtual int [numberOfIdenticalNodesFamilies](#) ([ConnectionFamily](#) *) const
finds the number of node families that are common between the two connections (the node families should be exactly the same, not isA)
- virtual QStringList [synonyms](#) (const QString &role) const
finds possible role synonyms by looking at child families and finding roles with the same index

Static Public Member Functions

- static [ConnectionFamily](#) * [cast](#) ([ItemFamily](#) *)
cast to connection family

Protected Member Functions

- virtual bool [isA](#) (int) const
indicates whether or not the given ID is this family or any of its parent families

Static Protected Member Functions

- static bool [checkRestrictions](#) (const QString &restriction, const QList< [NodeHandle](#) * > &, bool checkFull=true)
check for restrictions. RESTRICTIONS ARE HARD CODED. SEE FUNCTION CODE

Protected Attributes

- QList< [ConnectionFamily](#) * > [parentFamilies](#)
all the parents
- QList< [ConnectionFamily](#) * > [childFamilies](#)
all the families that are under this family
- QList< QPair< int, int > > [nodeRoles](#)
the role ID and type ID of each node that is involved in this connection

Static Protected Attributes

- static QHash< QString, int > [ROLEID](#)
stored a list of all possible node roles as IDs
- static QStringList [ALLROLENAMES](#)
all role names. used to assign role IDs

6.17.1 Detailed Description

This class defines the family of a connection. Inherits from [ItemFamily](#) It contains a list of [ConnectionGraphicsItems](#) that is the default for this family of connections.

6.17.2 Member Function Documentation

6.17.2.1 `bool Tinkercell::ConnectionFamily::addParticipant (const QString & role, const QString & family) [virtual]`

add a participant

in a connection and related functions

Parameters

QString role of participant

QString type of participant, must be a family name of a node

Returns

bool false if the participant family does not exist (i.e role not added)

6.17.2.2 `QList< ItemFamily * > Tinkercell::ConnectionFamily::findValidChildFamilies (const QList< NodeHandle * > & nodes, bool checkFull = true) [virtual]`

find child-families of this family that the given set of nodes can potentially belong with

Parameters

bool `QList<NodeHandle*>` node handles

bool use false here if the list of nodes is a partial list

Returns

`QList<ItemFamily*>` valid connection families

6.17.2.3 `bool Tinkercell::ConnectionFamily::isA (int id) const [protected, virtual]`

indicates whether or not the given ID is this family or any of its parent families

indicates whether or not the given string is the name of this family or any of its parent families

Reimplemented from [Tinkercell::ItemFamily](#).

6.17.2.4 `bool Tinkercell::ConnectionFamily::isValidSet (const QList< NodeHandle * > & nodes, bool checkFull = true) [virtual]`

checks if this family is compatible with a connection composed of the given set of nodes

Parameters

bool `QList<NodeHandle*>` node handles

bool use false here if the list of nodes is a partial list

Returns

Boolean

6.17.2.5 int TinkerCell::ConnectionFamily::numberOfIdenticalNodesFamilies (ConnectionFamily * *other*) const [virtual]

finds the number of node families that are common between the two connections (the node families should be exactly the same, not isA)

Parameters

ConnectionFamily *

Returns

bool

6.17.2.6 QString TinkerCell::ConnectionFamily::participantFamily (const QString & *role*) const [virtual]

get participant family

Parameters

QString role of participant

Returns

QString family name (empty if none)

6.17.2.7 QStringList TinkerCell::ConnectionFamily::participantRoles () const [virtual]

get all participant roles

Returns

QStringList role names (may not be unique)

6.17.2.8 QStringList TinkerCell::ConnectionFamily::participantTypes () const [virtual]

get all participant family names

Returns

QStringList family names (may not be unique)

6.17.2.9 QStringList TinkerCell::ConnectionFamily::synonyms (const QString & role) const [virtual]

finds possible role synonyms by looking at child families and finding roles with the same index

Parameters

QString role name

Returns

QStringList synonyms for the input role

The documentation for this class was generated from the following files:

- ItemFamily.h
- ItemFamily.cpp

6.18 TinkerCell::ConnectionGraphicsItem Class Reference

A graphics nodes item that draws connection between two or more nodes and the arrow heads at the ends.

```
#include <ConnectionGraphicsItem.h>
```

Classes

- class [ControlPoint](#)
A control point with a pointer to a [ConnectionGraphicsItem](#).
- class [CurveSegment](#)
A set of control points and two arrow heads.

Public Types

- enum [LineType](#) { **line**, **bezier** }
line or bezier
- enum { **Type** = UserType + 5 }
for enabling dynamic_cast

Public Member Functions

- [ConnectionGraphicsItem](#) (QGraphicsItem *parent=0)
- [ConnectionGraphicsItem](#) (const QList< [NodeGraphicsItem](#) * > &, const QList< [NodeGraphicsItem](#) * > &, QGraphicsItem *parent=0)
- [ConnectionGraphicsItem](#) (const [ConnectionGraphicsItem](#) ©)
- virtual [ConnectionGraphicsItem](#) & operator= (const [ConnectionGraphicsItem](#) ©)
- virtual [ConnectionGraphicsItem](#) & copyPoints (const [ConnectionGraphicsItem](#) ©)

- virtual [ConnectionGraphicsItem](#) * [clone](#) () const
make a copy of this connection item
- virtual bool [isValid](#) ()
returns the bounding rectangle for this reaction figure
- virtual [ItemHandle](#) * [handle](#) () const
get the handle of this connection
- virtual void [setHandle](#) ([ItemHandle](#) *)
set the handle of this connection
- virtual QList< [ControlPoint](#) * > [controlPoints](#) (bool includeEnds=false) const
list of pointers to all the control points
- virtual QList< [QGraphicsItem](#) * > [controlPointsAsGraphicsItems](#) (bool includeEnds=false) const
list of pointers to all the control points
- virtual [QPainterPath](#) [shape](#) () const
gets a path that represents this reaction
- virtual void [setPath](#) (const [QPainterPath](#) &path)
set the path for this connection
- virtual void [clear](#) (bool all=false)
Clear all shapes and control points.
- virtual void [refresh](#) (bool arrows=true)
refresh the path if any controlpoints have moved
- virtual void [setPen](#) ([QPen](#) pen, bool permanently=false)
set the color and line width for drawing this connection
- virtual [QPen](#) [pen](#) () const
get the pen currently being used to draw this connection
- virtual void [setControlPointsVisible](#) (bool visible=true)
set visibility of control points
- void [showControlPoints](#) ()
show control points. same as setControlPointsVisible(true)
- void [hideControlPoints](#) ()
hide control points. same as setControlPointsVisible(false)
- virtual bool [isModifier](#) () const
check is this connection represents a modifier, i.e. points to the centerRegion of another connection
- virtual QList< [NodeGraphicsItem](#) * > [nodes](#) () const
get all nodes that are connected

- virtual QList< [NodeGraphicsItem](#) * > [nodesWithArrows](#) () const
get all nodes that have an arrow pointing to them
- virtual QList< [NodeGraphicsItem](#) * > [nodesWithoutArrows](#) () const
get all nodes that do NOT have an arrow pointing to them
- virtual QList< [NodeGraphicsItem](#) * > [nodesDisconnected](#) () const
get all nodes that are not directle connected to the main connection, such as modifier nodes
- virtual QList< [QGraphicsItem](#) * > [nodesAsGraphicsItems](#) () const
get all nodes that are connected
- virtual QList< [ArrowHeadItem](#) * > [arrowHeads](#) () const
get all the arrowHeads associated with the nodes. The order is the same order as [nodes\(\)](#), so values can be 0
- virtual QList< [QGraphicsItem](#) * > [arrowHeadsAsGraphicsItems](#) () const
get all the arrowHeads associated with the nodes The order is the same order as [nodes\(\)](#), so values can be 0
- virtual QList< [ArrowHeadItem](#) * > [modifierArrowHeads](#) () const
get all the arrowHeads NOT associated with the nodes
- virtual [NodeGraphicsItem](#) * [nodeAt](#) (int index) const
get the node that connected to the particular path
- virtual int [indexOf](#) ([QGraphicsItem](#) *node) const
get the index of the node
- virtual void [replaceNodeAt](#) (int, [NodeGraphicsItem](#) *)
replace the node at the particular position with a new node
- virtual void [replaceNode](#) ([NodeGraphicsItem](#) *, [NodeGraphicsItem](#) *)
replace one node in the reaction with another
- virtual [ArrowHeadItem](#) * [arrowAt](#) (int index) const
get the arrow head at the particular index
- virtual [ArrowHeadItem](#) * [modifierArrowAt](#) (int index) const
get the modifier arrow head at the particular index
- virtual [~ConnectionGraphicsItem](#) ()
- virtual qreal [slopeAtPoint](#) (const [QPointF](#) &point)
get slope at the given point (or closest point)
- virtual [ControlPoint](#) * [centerPoint](#) () const
the center point (if one exists)
- virtual [QPointF](#) [centerLocation](#) () const
the center point (if one exists)

- virtual QRectF [boundingRect](#) () const
bounding rect
- virtual QRectF [sceneBoundingRect](#) () const
scene bounding rect
- virtual int [type](#) () const
for enabling dynamic_cast

Static Public Member Functions

- static [ConnectionGraphicsItem](#) * [cast](#) (QGraphicsItem *)
cast a graphics item to a connection graphics item using qgraphicsitem_cast
- static QList< [ConnectionGraphicsItem](#) * > [cast](#) (const QList< QGraphicsItem * > &)
cast a list of graphics item to a list of connection graphics items using qgraphicsitem_cast
- static [ConnectionGraphicsItem](#) * [topLevelConnectionItem](#) (QGraphicsItem *item, bool includeControlPoints=false)
gets the connection graphics item from its child item

Public Attributes

- QString [name](#)
just a name used identifying the connection
- QString [className](#)
used for checking type before static casts
- QPen [defaultPen](#)
permanent pen for this control point
- QString [groupID](#)
for identifying which scene this item belongs in
- [LineType](#) [lineType](#)
type of line for this reaction - line or bezier
- QList< [CurveSegment](#) > [curveSegments](#)
vector of vector of control point
- qreal [arrowHeadDistance](#)
distance from arrow head to the item that it is connected to
- bool [controlPointsVisible](#)
indicates whether to show lines around the curves

- QSizeF [centerRegion](#)
a rectangle that sits at the center of the connector
- [ArrowHeadItem](#) * [centerRegionItem](#)
the image on the rectangle that sits at the center of the connector

Static Public Attributes

- static const QString [CLASSNAME](#) = QString("ConnectionGraphicsItem")
used for checking type before static casts
- static QString [DefaultMiddleItemFile](#)
used to initialize the middle item for a connection
- static QString [DefaultArrowHeadFile](#)
used to initialize the arrow heads for a connection
- static const int [numLineTypes](#) = 2
number of different type of shapes available

Protected Member Functions

- virtual void [refreshBoundaryPath](#) ()
update the boundary path
- virtual void [adjustEndPoints](#) (bool arrows=true)
adjust the end control points so that they point straight

Protected Attributes

- [ItemHandle](#) * [itemHandle](#)
Tinkercell object that this drawable belongs in.
- QGraphicsPathItem * [boundaryPathItem](#)
path for drawing the boundary region
- QGraphicsPathItem * [outerPathItem](#)
path of the outline (usually white)
- QGraphicsPathItem * [mainPathItem](#)
path of the main curve
- QPainterPath [pathShape](#)
path of the selection region of the entire connection

- QRectF [pathBoundingRect](#)

the boundary rectangle for this path. It is recomputed during each refresh.

6.18.1 Detailed Description

A graphics nodes item that draws connection between two or more nodes and the arrow heads at the ends.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 TinkerCell::ConnectionGraphicsItem::ConnectionGraphicsItem (QGraphicsItem * *parent* = 0)

Constructor: does nothing

Constructor: initialize everything

6.18.2.2 TinkerCell::ConnectionGraphicsItem::ConnectionGraphicsItem (const QList< NodeGraphicsItem * > & *from*, const QList< NodeGraphicsItem * > & *to*, QGraphicsItem * *parent* = 0)

Constructor: constructs linear curve segments with arrow heads on the second set of nodes

Parameters

QList<NodeGraphicsItem>* list of nodes to connect from (no arrow heads)

QList<NodeGraphicsItem>* list of nodes to connect to (have arrow heads)

6.18.2.3 TinkerCell::ConnectionGraphicsItem::ConnectionGraphicsItem (const ConnectionGraphicsItem & *copy*)

Copy Constructor: copies handle but not control points

Copy Constructor: deep copy of all pointers

6.18.2.4 TinkerCell::ConnectionGraphicsItem::~~ConnectionGraphicsItem () [virtual]

Destructor: deletes all control points

Destructor: deletes all shapes and control points

6.18.3 Member Function Documentation

6.18.3.1 void TinkerCell::ConnectionGraphicsItem::adjustEndPoints (bool *arrowTransform* = true) [protected, virtual]

adjust the end control points so that they point straight

Parameters

bool adjust arrow transformations

void

Returns

void

6.18.3.2 ArrowHeadItem * Tinkercell::ConnectionGraphicsItem::arrowAt (int *index*) const [virtual]

get the arrow head at the particular index

find the arrow head at the particular index

Parameters

index less than size of curveSegments

Returns

node item or 0

6.18.3.3 QList< ArrowHeadItem * > Tinkercell::ConnectionGraphicsItem::arrowHeads () const [virtual]

get all the arrowHeads associated with the nodes. The order is the same order as [nodes\(\)](#), so values can be 0

get all the arrow heads in the same order as nodes

Returns

node item list

6.18.3.4 QList< QGraphicsItem * > Tinkercell::ConnectionGraphicsItem::arrowHeadsAsGraphicsItems () const [virtual]

get all the arrowHeads associated with the nodes The order is the same order as [nodes\(\)](#), so values can be 0

get all the arrow heads in the same order as nodes

Returns

arrow item list

node item list

6.18.3.5 QList< ConnectionGraphicsItem * > Tinkercell::ConnectionGraphicsItem::cast (const QList< QGraphicsItem * > & *list*) [static]

cast a list of graphics item to a list of connection graphics items using qgraphicsitem_cast

Parameters

QList<QGraphicsItem>* graphics items

Returns

QList<ConnectionGraphicsItem>* can be empty if no cast is invalid

6.18.3.6 **ConnectionGraphicsItem * TinkerCell::ConnectionGraphicsItem::cast (QGraphicsItem * q) [static]**

cast a graphics item to a connection graphics item using `qgraphicsitem_cast`

Parameters

*QGraphicsItem** graphics item

Returns

*ConnectionGraphicsItem** can be 0 if the cast is invalid

6.18.3.7 **QPointF TinkerCell::ConnectionGraphicsItem::centerLocation () const [virtual]**

the center point (if one exists)

the center location

6.18.3.8 **void TinkerCell::ConnectionGraphicsItem::clear (bool all = false) [virtual]**

Clear all shapes and control points.

Parameters

void

Returns

void

6.18.3.9 **ConnectionGraphicsItem * TinkerCell::ConnectionGraphicsItem::clone () const [virtual]**

make a copy of this connection item

make a copy of this item

6.18.3.10 **ConnectionGraphicsItem & TinkerCell::ConnectionGraphicsItem::copyPoints (const ConnectionGraphicsItem & copy) [virtual]**

operator =: copy just the control point positions and pen

6.18.3.11 void Tinkercell::ConnectionGraphicsItem::hideControlPoints ()

hide control points. same as setControlPointsVisible(false)

Returns

void

6.18.3.12 int Tinkercell::ConnectionGraphicsItem::indexOf (QGraphicsItem * *target*) const [virtual]

get the index of the node

find the index of the node

Parameters

node in this connection

Returns

index, -1 if node not found

6.18.3.13 bool Tinkercell::ConnectionGraphicsItem::isModifier () const [virtual]

check is this connection represents a modifier, i.e. points to the centerRegion of another connection

Returns

boolean

6.18.3.14 bool Tinkercell::ConnectionGraphicsItem::isValid () [virtual]

returns the bounding rectangle for this reaction figure

checks that this is a valid drawable

paint method. Call's parent's after drawing boundary true

checks that this is a valid drawable

6.18.3.15 ArrowHeadItem * Tinkercell::ConnectionGraphicsItem::modifierArrowAt (int *index*) const [virtual]

get the modifier arrow head at the particular index

find the modifier arrow head at the particular index

Parameters

index less than size of curveSegments

Returns

node item or 0

6.18.3.16 `QList< ArrowHeadItem * > TinkerCell::ConnectionGraphicsItem::modifierArrowHeads () const`
[virtual]

get all the arrowHeads NOT associated with the nodes

find all the modifier arrow heads in the same order as nodes

Returns

graphics item list
node item list

6.18.3.17 `NodeGraphicsItem * TinkerCell::ConnectionGraphicsItem::nodeAt (int index) const`
[virtual]

get the node that connected to the particular path

find the node that connected to the particular path

Parameters

index less than size of curveSegments

Returns

node item or 0

6.18.3.18 `QList< NodeGraphicsItem * > TinkerCell::ConnectionGraphicsItem::nodes () const`
[virtual]

get all nodes that are connected

find all the nodes that are connected

Returns

node item list
node item list or 0

6.18.3.19 `QList< QGraphicsItem * > TinkerCell::ConnectionGraphicsItem::nodesAsGraphicsItems () const`
[virtual]

get all nodes that are connected

find all the nodes that are connected

Returns

graphics item list
node item list or 0

6.18.3.20 `QList< NodeGraphicsItem * > Tinkercell::ConnectionGraphicsItem::nodesDisconnected () const`
[virtual]

get all nodes that are not directle connected to the main connection, such as modifier nodes

find all the nodes that are connected

Returns

node item list

node item list or 0

6.18.3.21 `QList< NodeGraphicsItem * > Tinkercell::ConnectionGraphicsItem::nodesWithArrows () const`
[virtual]

get all nodes that have an arrow pointing to them

find all the nodes that are connected

Returns

node item list

node item list or 0

6.18.3.22 `QList< NodeGraphicsItem * > Tinkercell::ConnectionGraphicsItem::nodesWithoutArrows () const`
[virtual]

get all nodes that do NOT have an arrow pointing to them

find all the nodes that are connected

Returns

node item list

node item list or 0

6.18.3.23 `ConnectionGraphicsItem & Tinkercell::ConnectionGraphicsItem::operator= (const ConnectionGraphicsItem & copy)` **[virtual]**

operator =: remove everything from original connection and copy everything from the given connection

operator =: copy just the control point positions and pen

6.18.3.24 `QPen Tinkercell::ConnectionGraphicsItem::pen () const` **[virtual]**

get the pen currently being used to draw this connection

Returns

QPen pen

**6.18.3.25 void TinkerCell::ConnectionGraphicsItem::refresh (bool *arrowTransform* = *true*)
[virtual]**

refresh the path if any controlpoints have moved

Parameters

bool tranform arrow heads

Returns

void

Parameters

void

Returns

void

**6.18.3.26 void TinkerCell::ConnectionGraphicsItem::replaceNode (NodeGraphicsItem *
oldNode, NodeGraphicsItem * *newNode*) [virtual]**

replace one node in the reaction with another

Parameters

target node to replace

new node

Returns

void

**6.18.3.27 void TinkerCell::ConnectionGraphicsItem::replaceNodeAt (int *index*,
NodeGraphicsItem * *nodeItem*) [virtual]**

replace the node at the particular position with a new node

Parameters

index where to insert the new node

new node

Returns

void

6.18.3.28 void Tinkercell::ConnectionGraphicsItem::setControlPointsVisible (bool *visible* = *true*) [virtual]

set visibility of control points

Parameters

visible = true, invisible = false

Returns

void

6.18.3.29 void Tinkercell::ConnectionGraphicsItem::setPath (const QPainterPath & *path*) [virtual]

set the path for this connection

Parameters

QPainterPath *path*

Returns

void

6.18.3.30 void Tinkercell::ConnectionGraphicsItem::setPen (QPen *pen*, bool *permanently* = *false*) [virtual]

set the color and line width for drawing this connection

Parameters

QPen *pen*

bool also set the default pen?

Returns

void

6.18.3.31 QPainterPath Tinkercell::ConnectionGraphicsItem::shape () const [virtual]

gets a path that represents this reaction

gets a path that is constructed by uniting all the shape paths

6.18.3.32 void Tinkercell::ConnectionGraphicsItem::showControlPoints ()

show control points. same as setControlPointsVisible(true)

Returns

void

6.18.3.33 qreal TinkerCell::ConnectionGraphicsItem::slopeAtPoint (const QPointF & *point*) [virtual]

get slope at the given point (or closest point)

find slope at the given point (or closest point)

6.18.3.34 ConnectionGraphicsItem * TinkerCell::ConnectionGraphicsItem::topLevelConnectionItem (QGraphicsItem * *item*, bool *includeControlPoints* = false) [static]

gets the connection graphics item from its child item

Parameters

*QGraphicsItem** the target item

bool using true here will return the connection item for a control point, otherwise control points are ignored

The documentation for this class was generated from the following files:

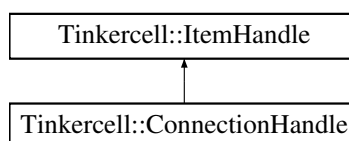
- ConnectionGraphicsItem.h
- ConnectionGraphicsItem.cpp

6.19 TinkerCell::ConnectionHandle Class Reference

The handles are used to bring together data and graphics items. Connection Handle contains pointers to all the graphics items that belong to it, the tools that apply to this item, the data for this item, the family that it belongs with, and pointers to nodes connected (in and out).

```
#include <ItemHandle.h>
```

Inheritance diagram for TinkerCell::ConnectionHandle:



Public Member Functions

- virtual QList< [NodeHandle](#) * > [nodes](#) (int role=0) const
returns all the nodes connected to all the connectors in this handle
- virtual void [addNode](#) ([NodeHandle](#) *, int role=0)
add a node to this connection (only applies to connections with NO graphics items)
- virtual void [clearNodes](#) ()
clear all nodes in connection (only applies to connections with NO graphics items)

- virtual `QList< NodeHandle * > nodesIn ()` const
returns all the nodes that are on the "input" side of this connection. If this connection is represented by graphics items, then this is determined by looking at which nodes have an arrow-head associated with them in graphics items. If there are no graphics items, then this function uses the `_nodes` list to find the "in" nodes (role = -1).
- virtual `QList< NodeHandle * > nodesOut ()` const
If this connection is represented by graphics items, then this is determined by looking at which nodes have NO arrow-head associated with them in graphics items. If there are no graphics items, then this function uses the `_nodes` list to find the "out" nodes (role = +1).
- [ConnectionHandle](#) (const `QString` &`name`=`QString()`, [ConnectionFamily](#) *`family`=0)
default constructor -- initializes everything
- [ConnectionHandle](#) ([ConnectionFamily](#) *`family`, const `QString` &`name`=`QString()`)
one parameter constructor -- initializes everything
- [ConnectionHandle](#) (const [ConnectionHandle](#) &)
copy constructor -- deep copy of data, but shallow copy of graphics items
- virtual [ConnectionHandle](#) & `operator=` (const [ConnectionHandle](#) &)
operator =
- [ConnectionHandle](#) ([ConnectionFamily](#) *`family`, [ConnectionGraphicsItem](#) *`item`)
two parameter constructor
- virtual void `setFamily` ([ItemFamily](#) *`family`, bool `useCommand`=true)
set the family for this handle
- virtual [ItemHandle](#) * `clone ()` const
clone of this handle
- virtual [ItemFamily](#) * `family ()` const
family for this handle
- virtual `QList< ItemFamily * > findValidChildFamilies ()` const
find child-families of the current family that this connection can potentially belong with

Static Public Member Functions

- static [ConnectionHandle](#) * `cast` ([ItemHandle](#) *)
checks if the item handle is a connection handle and casts it as a connection item. Returns 0 if it is not a node item
- static `QList< ConnectionHandle * > cast` (const `QList< ItemHandle * > &`)
checks if the item handles are connection handles and casts them as connection items. Returns `QList<ConnectionHandle>`*

Public Attributes

- [ConnectionFamily](#) * [connectionFamily](#)
the family for this connection handle
- QList< QPair< [NodeHandle](#) *, int > > [nodesWithRoles](#)
the nodes that are connected by this connection and the role of each node. this list is ONLY used for connections with NO graphics items -1 and 1 are reserved roles, indicating in and out nodes

Static Public Attributes

- static const int [TYPE](#) = 2
this number is used to identify when an item handle is a connection handle

6.19.1 Detailed Description

The handles are used to bring together data and graphics items. Connection Handle contains pointers to all the graphics items that belong to it, the tools that apply to this item, the data for this item, the family that it belongs with, and pointers to nodes connected (in and out).

6.19.2 Constructor & Destructor Documentation

6.19.2.1 TinkerCell::ConnectionHandle::ConnectionHandle (ConnectionFamily * *family*, const QString & *name* = QString())

one parameter constructor -- initializes everything

Parameters

*ConnectionFamily** connection family
QString name

6.19.2.2 TinkerCell::ConnectionHandle::ConnectionHandle (ConnectionFamily * *family*, ConnectionGraphicsItem * *item*)

two parameter constructor

Parameters

*ConnectionFamily** initial family
*ConnectionGraphicsItem** connection graphics item

6.19.3 Member Function Documentation

6.19.3.1 void TinkerCell::ConnectionHandle::addNode (NodeHandle * *h*, int *role* = 0) [virtual]

add a node to this connection (only applies to connections with NO graphics items)

Parameters

*NodeHandle** node

int role of this node. -1 is for "in" nodes. +1 is for "out" nodes. Use any other values for specific purposes

6.19.3.2 **ConnectionHandle * Tinkercell::ConnectionHandle::cast (ItemHandle * *item*) [static]**

checks if the item handle is a connection handle and casts it as a connection item. Returns 0 if it is not a node item

Parameters

*ItemHandle** item

6.19.3.3 **QList< ConnectionHandle * > Tinkercell::ConnectionHandle::cast (const QList< ItemHandle * > & *items*) [static]**

checks if the item handles are connection handles and casts then as connection items. Returns QList<ConnectionHandle*>

Parameters

Returns QList<ItemHandle*> items

6.19.3.4 **ItemHandle * Tinkercell::ConnectionHandle::clone () const [virtual]**

clone of this handle

Returns

ItemFamily* connection handle as item handle

Reimplemented from [Tinkercell::ItemHandle](#).

6.19.3.5 **ItemFamily * Tinkercell::ConnectionHandle::family () const [virtual]**

family for this handle

Returns

ItemFamily* connection family as item family

Reimplemented from [Tinkercell::ItemHandle](#).

6.19.3.6 `QList< ItemFamily * > Tinkercell::ConnectionHandle::findValidChildFamilies () const [virtual]`

find child-families of the current family that this connection can potentially belong with

Returns

`QList<ItemFamily*>` valid connection families

6.19.3.7 `QList< NodeHandle * > Tinkercell::ConnectionHandle::nodes (int role = 0) const [virtual]`

returns all the nodes connected to all the connectors in this handle

Returns

`QList<NodeHandle*>` list of node handles

6.19.3.8 `QList< NodeHandle * > Tinkercell::ConnectionHandle::nodesIn () const [virtual]`

returns all the nodes that are on the "input" side of this connection. If this connection is represented by graphics items, then this is determined by looking at which nodes have an arrow-head associated with them in graphics items. If there are no graphics items, then this function uses the `_nodes` list to find the "in" nodes (role = -1).

Returns

`QList<NodeHandle*>` list of node handles

6.19.3.9 `QList< NodeHandle * > Tinkercell::ConnectionHandle::nodesOut () const [virtual]`

If this connection is represented by graphics items, then this is determined by looking at which nodes have NO arrow-head associated with them in graphics items. If there are no graphics items, then this function uses the `_nodes` list to find the "out" nodes (role = +1).

Returns

`QList<NodeHandle*>` list of node handles

6.19.3.10 `void Tinkercell::ConnectionHandle::setFamily (ItemFamily * family, bool useCommand = true) [virtual]`

set the family for this handle

Parameters

*ConnectionFamily** connection family

Reimplemented from [Tinkercell::ItemHandle](#).

The documentation for this class was generated from the following files:

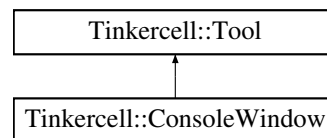
- ItemHandle.h
- ItemHandle.cpp

6.20 Tinkercell::ConsoleWindow Class Reference

Used to create an output window that can display outputs.

```
#include <ConsoleWindow.h>
```

Inheritance diagram for Tinkercell::ConsoleWindow:



Public Slots

- virtual void [eval](#) (const QString &)
send a command to the console window to be evaluated
- virtual void [message](#) (const QString &)
print a message in the output window
- virtual void [error](#) (const QString &)
print an error message in the output window
- virtual void [printTable](#) (const [DataTable](#)< qreal > &dataTable)
print a data table (tab-delimited) in the output window
- virtual void [clear](#) ()
clear the output window
- virtual void [freeze](#) ()
freeze the output window. Frozen window will not be responsive to commands
- virtual void [unfreeze](#) ()
unfreeze the output window. Frozen window will not be responsive to commands

Signals

- void [commandExecuted](#) (const QString &command)
the user requested to execute the given command

- void [commandInterrupted](#) ()
the user requested to interrupt the current process

Public Member Functions

- [ConsoleWindow](#) ([MainWindow](#) *main=0)
constructor -- initialize main window
- virtual [CommandTextEdit](#) * [editor](#) ()
the command window's editor
- virtual void [setInterpreter](#) ([InterpreterThread](#) *)
set the interpreter for the console window, e.g. new [PythonInterpreterThread](#)

Static Public Attributes

- static [QString](#) [Prompt](#)
the string used at the prompt
- static [QColor](#) [BackgroundColor](#) = [QColor](#)("#000000")
the background color for console
- static [QColor](#) [PlainTextColor](#) = [QColor](#)("#FEFFEC")
the font color for plain text
- static [QColor](#) [ErrorTextColor](#) = [QColor](#)("#FF6F0F")
the font color for error messages
- static [QColor](#) [OutputTextColor](#) = [QColor](#)("#33FF00")
the font color for outputs
- static [QColor](#) [TableTextColor](#) = [QColor](#)("#FFFF00")
the font color for table headers

Protected Attributes

- [CommandTextEdit](#) [commandTextEdit](#)
the command window
- [InterpreterThread](#) * [interpreter](#)
the optional interpreter for processing commands

6.20.1 Detailed Description

Used to create an output window that can display outputs.

6.20.2 Member Function Documentation

6.20.2.1 void Tinkercell::ConsoleWindow::message (const QString & s) [virtual, slot]

print a message in the output window

show a message text in the output window

The documentation for this class was generated from the following files:

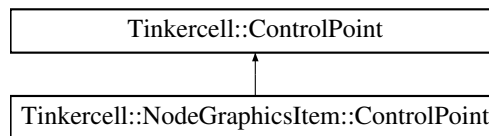
- ConsoleWindow.h
- ConsoleWindow.cpp

6.21 Tinkercell::NodeGraphicsItem::ControlPoint Class Reference

a control point with a pointer to a [NodeGraphicsItem](#)

```
#include <NodeGraphicsItem.h>
```

Inheritance diagram for Tinkercell::NodeGraphicsItem::ControlPoint:



Public Types

- enum { **Type** = UserType + 2 }
for enabling dynamic_cast

Public Member Functions

- [ControlPoint](#) ([NodeGraphicsItem](#) *drawable_ptr=0, QGraphicsItem *parent=0)
Constructor: Setup colors and z value.
- [ControlPoint](#) (const [ControlPoint](#) ©)
Copy Constructor.
- virtual [ControlPoint](#) & operator= (const [ControlPoint](#) ©)
operator =
- virtual [Tinkercell::ControlPoint](#) * clone () const
make a copy of this control point

- virtual int [type](#) () const
for enabling dynamic_cast
- virtual void [sideEffect](#) ()
side effect when moved. always call this after moving
- virtual void [paint](#) (QPainter *painter, const QStyleOptionGraphicsItem *option=new QStyleOptionGraphicsItem(), QWidget *widget=0)
paint method.
- virtual [ItemHandle](#) * [handle](#) () const
same as nodeItem->handle()
- virtual void [setHandle](#) ([ItemHandle](#) *)
set the nodeItem->setHandle(..)
- [~ControlPoint](#) ()
destructor

Public Attributes

- [NodeGraphicsItem](#) * [nodeItem](#)
idrawables that this control point belong in

6.21.1 Detailed Description

a control point with a pointer to a [NodeGraphicsItem](#)

6.21.2 Member Function Documentation

6.21.2.1 TinkerCell::ControlPoint * TinkerCell::NodeGraphicsItem::ControlPoint::clone () const [virtual]

make a copy of this control point

make a copy of this item

Reimplemented from [TinkerCell::ControlPoint](#).

6.21.2.2 NodeGraphicsItem::ControlPoint & TinkerCell::NodeGraphicsItem::ControlPoint::operator= (const ControlPoint & copy) [virtual]

operator =

Copy operator

6.21.2.3 void TinkerCell::NodeGraphicsItem::ControlPoint::paint (QPainter * *painter*, const QStyleOptionGraphicsItem * *option* = new QStyleOptionGraphicsItem(), QWidget * *widget* = 0) [virtual]

paint method.

paint method. Call's parent's

Reimplemented from [TinkerCell::ControlPoint](#).

The documentation for this class was generated from the following files:

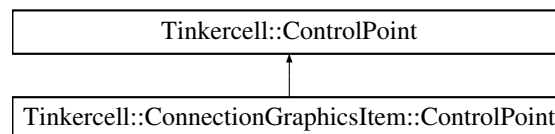
- NodeGraphicsItem.h
- NodeGraphicsItem.cpp

6.22 TinkerCell::ConnectionGraphicsItem::ControlPoint Class Reference

A control point with a pointer to a [ConnectionGraphicsItem](#).

```
#include <ConnectionGraphicsItem.h>
```

Inheritance diagram for TinkerCell::ConnectionGraphicsItem::ControlPoint:



Public Types

- enum { **Type** = UserType + 7 }
for enabling dynamic_cast

Public Member Functions

- [ControlPoint](#) (ConnectionGraphicsItem *reaction_ptr=0, QGraphicsItem *parent=0)
Constructor: Setup colors and z value.
- [ControlPoint](#) (const QPointF &pos, [ConnectionGraphicsItem](#) *reaction_ptr=0, QGraphicsItem *parent=0)
Constructor: constructor with position.
- [ControlPoint](#) (const [ControlPoint](#) ©)
Copy Constructor.
- virtual [ControlPoint](#) & operator= (const [ControlPoint](#) ©)
operator =

- virtual int [type](#) () const
for enabling dynamic_cast
- [~ControlPoint](#) ()
destructor
- virtual [Tinkercell::ControlPoint](#) * [clone](#) () const
side effect when moved. always call this after moving
- virtual [ItemHandle](#) * [handle](#) () const
same as connectionItem->handle()
- virtual void [setHandle](#) ([ItemHandle](#) *)
same as connectionItem->setHandle(...)

Public Attributes

- [ConnectionGraphicsItem](#) * [connectionItem](#)
idrawables that this control point belong in

6.22.1 Detailed Description

A control point with a pointer to a [ConnectionGraphicsItem](#).

6.22.2 Constructor & Destructor Documentation

6.22.2.1 Tinkercell::ConnectionGraphicsItem::ControlPoint::~~ControlPoint ()

destructor

destructor

6.22.3 Member Function Documentation

6.22.3.1 ControlPoint * Tinkercell::ConnectionGraphicsItem::ControlPoint::clone () const [virtual]

side effect when moved. always call this after moving

make a copy of this item

make a copy of this control point

Reimplemented from [Tinkercell::ControlPoint](#).

6.22.3.2 ConnectionGraphicsItem::ControlPoint & Tinker- cell::ConnectionGraphicsItem::ControlPoint::operator= (const ControlPoint & copy) [virtual]

operator =

Copy operator

The documentation for this class was generated from the following files:

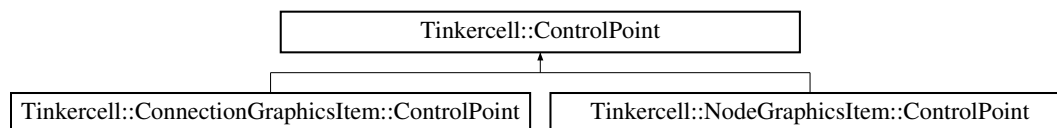
- ConnectionGraphicsItem.h
- ConnectionGraphicsItem.cpp

6.23 TinkerCell::ControlPoint Class Reference

A simple circle or square that is used for changing specific locations.

```
#include <ControlPoint.h>
```

Inheritance diagram for TinkerCell::ControlPoint:



Public Types

- enum { **Type** = UserType + 1 }
paint method. Call's parent's paint after setting antialiasing to true
- enum ShapeType { circle, square, triangle }
type of shape to paint.

Public Member Functions

- virtual qreal x ()
x position
- virtual qreal y ()
y position
- ControlPoint (QGraphicsItem *parent=0)
Constructor: Setup colors and z value.
- ControlPoint (const ControlPoint ©)
copy constructor
- virtual int type () const

for enabling dynamic_cast

- virtual void [sideEffect](#) ()
side effect when moved. always call this after moving
- virtual [ControlPoint](#) * [clone](#) () const
make a copy of this control point
- virtual void [paint](#) (QPainter *painter, const QStyleOptionGraphicsItem *option=new QStyleOptionGraphicsItem(), QWidget *widget=0)
paint method.
- virtual QRectF [boundingRect](#) () const
bounding rect method.
- virtual void [setRect](#) (const QRectF &)
set size rect.
- virtual QRectF [rect](#) () const
get size rect.
- virtual [ItemHandle](#) * [handle](#) () const
get the handle of this control point, usually 0 or the parent's handle
- virtual void [setHandle](#) ([ItemHandle](#) *)
set the handle of this control point, usually sets parent's handle or does nothing

Static Public Member Functions

- static [ControlPoint](#) * [cast](#) (QGraphicsItem *item)
Gets the control point item from one of its child items.

Public Attributes

- QBrush [defaultBrush](#)
permanent brush for this control point
- QPen [defaultPen](#)
permanent pen for this control point
- QSizeF [defaultSize](#)
default size for this item
- [ShapeType](#) [shapeType](#)
type of shape to paint.

Protected Attributes

- `QRectF` bounds

6.23.1 Detailed Description

A simple circle or square that is used for changing specific locations.

6.23.2 Member Enumeration Documentation

6.23.2.1 anonymous enum

paint method. Call's parent's paint after setting antialiasing to true for enabling `dynamic_cast`

6.23.3 Constructor & Destructor Documentation

6.23.3.1 `Tinkercell::ControlPoint::ControlPoint (const ControlPoint & copy)`

copy constructor

Copy Constructor.

6.23.4 Member Function Documentation

6.23.4.1 `ControlPoint * Tinkercell::ControlPoint::clone () const [virtual]`

make a copy of this control point

make a copy of this item

Reimplemented in [Tinkercell::ConnectionGraphicsItem::ControlPoint](#), and [Tinkercell::NodeGraphicsItem::ControlPoint](#).

6.23.4.2 `void Tinkercell::ControlPoint::paint (QPainter * painter, const QStyleOptionGraphicsItem * option = new QStyleOptionGraphicsItem(), QWidget * widget = 0) [virtual]`

paint method.

paint method. draw one of the shapes

Reimplemented in [Tinkercell::NodeGraphicsItem::ControlPoint](#).

6.23.4.3 `QRectF Tinkercell::ControlPoint::rect () const [virtual]`

get size rect.

bounding rect method.

6.23.4.4 void Tinkercell::ControlPoint::setRect (const QRectF & rect) [virtual]

set size rect.

set size.

The documentation for this class was generated from the following files:

- ControlPoint.h
- ConnectionGraphicsItem.cpp
- ControlPoint.cpp

6.24 Tinkercell::Core_FtoS Class Reference

Function to Signal converter for [MainWindow](#).

```
#include <C_API_Slots.h>
```

Signals

- void **allItems** (QSemaphore *, QList< [ItemHandle](#) * > *)
- void **selectedItems** (QSemaphore *, QList< [ItemHandle](#) * > *)
- void **itemsOffFamily** (QSemaphore *, QList< [ItemHandle](#) * > *, const QString &)
- void **itemsOffFamily** (QSemaphore *, QList< [ItemHandle](#) * > *, const QList< [ItemHandle](#) * > &, const QString &)
- void **find** (QSemaphore *, [ItemHandle](#) **, const QString &)
- void **findItems** (QSemaphore *, QList< [ItemHandle](#) * > *, const QStringList &)
- void **select** (QSemaphore *, [ItemHandle](#) *)
- void **deselect** (QSemaphore *)
- void **removeItem** (QSemaphore *, [ItemHandle](#) *)
- void **setPos** (QSemaphore *, [ItemHandle](#) *, qreal, qreal)
- void **setPos** (QSemaphore *, const QList< [ItemHandle](#) * > &, [DataTable](#)< qreal > &)
- void **getPos** (QSemaphore *, const QList< [ItemHandle](#) * > &, [DataTable](#)< qreal > *)
- void **getY** (QSemaphore *, qreal *, [ItemHandle](#) *)
- void **getX** (QSemaphore *, qreal *, [ItemHandle](#) *)
- void **moveSelected** (QSemaphore *, qreal, qreal)
- void **getFamily** (QSemaphore *, QString *, [ItemHandle](#) *)
- void **getName** (QSemaphore *, QString *, [ItemHandle](#) *)
- void **getUniqueName** (QSemaphore *, QString *, [ItemHandle](#) *)
- void **setName** (QSemaphore *, [ItemHandle](#) *, const QString &)
- void **getNames** (QSemaphore *, QStringList *, const QList< [ItemHandle](#) * > &)
- void **getUniqueNames** (QSemaphore *, QStringList *, const QList< [ItemHandle](#) * > &)
- void **isA** (QSemaphore *, int *, [ItemHandle](#) *, const QString &)
- void **outputText** (QSemaphore *, const QString &)
- void **errorReport** (QSemaphore *, const QString &)
- void **printFile** (QSemaphore *, const QString &)
- void **clearText** (QSemaphore *)
- void **outputTable** (QSemaphore *, const [DataTable](#)< qreal > &)
- void **createInputWindow** (QSemaphore *, const [DataTable](#)< qreal > &, const QString &, const QString &)

- void **createInputWindow** (QSemaphore *, long, const DataTable< qreal > &, const QString &, MatrixInputFunction)
- void **createSliders** (QSemaphore *, CThread *, const DataTable< qreal > &, MatrixInputFunction)
- void **addInputWindowOptions** (QSemaphore *, const QString &, int i, int j, const QStringList &)
- void **addInputWindowCheckbox** (QSemaphore *, const QString &, int i, int j)
- void **openNewWindow** (QSemaphore *, const QString &)
- void **isWindows** (QSemaphore *, int *)
- void **isMac** (QSemaphore *, int *)
- void **isLinux** (QSemaphore *, int *)
- void **appDir** (QSemaphore *, QString *)
- void **homeDir** (QSemaphore *, QString *)
- void **zoom** (QSemaphore *, qreal)
- void **getNumericalDataNames** (QSemaphore *, QStringList *, ItemHandle *)
- void **getTextDataNames** (QSemaphore *, QStringList *, ItemHandle *)
- void **getNumericalData** (QSemaphore *, DataTable< qreal > *, ItemHandle *, const QString &)
- void **setNumericalData** (QSemaphore *, ItemHandle *, const QString &, const DataTable< qreal > &)
- void **getTextData** (QSemaphore *, DataTable< QString > *, ItemHandle *, const QString &)
- void **setTextData** (QSemaphore *, ItemHandle *, const QString &, const DataTable< QString > &)
- void **getChildren** (QSemaphore *, QList< ItemHandle * > *, ItemHandle *)
- void **getParent** (QSemaphore *, ItemHandle **, ItemHandle *)
- void **getString** (QSemaphore *, QString *, const QString &)
- void **getFilename** (QSemaphore *, QString *)
- void **getSelectedString** (QSemaphore *, int *, const QString &, const QStringList &, const QString &)
- void **getNumber** (QSemaphore *, qreal *, const QString &)
- void **getNumbers** (QSemaphore *, const QStringList &, qreal *)
- void **askQuestion** (QSemaphore *, const QString &, int *)
- void **messageDialog** (QSemaphore *, const QString &)
- void **openFile** (QSemaphore *, const QString &)
- void **saveToFile** (QSemaphore *, const QString &)
- void **setSize** (QSemaphore *, ItemHandle *, double, double, int)
- void **getWidth** (QSemaphore *, ItemHandle *, double *)
- void **getHeight** (QSemaphore *, ItemHandle *, double *)
- void **setAngle** (QSemaphore *, ItemHandle *, double, int)
- void **getColor** (QSemaphore *, QString *, ItemHandle *)
- void **setColor** (QSemaphore *, ItemHandle *, const QString &, int)
- void **changeGraphics** (QSemaphore *, ItemHandle *, const QString &)
- void **changeArrowHead** (QSemaphore *, ItemHandle *, const QString &)
- void **screenshot** (QSemaphore *, const QString &, int, int)
- void **screenHeight** (QSemaphore *, int *)
- void **screenWidth** (QSemaphore *, int *)
- void **screenX** (QSemaphore *, int *)
- void **screenY** (QSemaphore *, int *)
- void **annotations** (QSemaphore *, QString *)
- void **insertAnnotation** (QSemaphore *, const QString &, double, double)
- void **setNumericalValues** (QSemaphore *, const NumericalDataTable &)
- void **setNumericalValue** (QSemaphore *, const QString &, double)

- void **setTextValues** (QSemaphore *, const [TextDataTable](#) &)
- void **setTextValue** (QSemaphore *, const QString &, const QString &)
- void **getNumericalValue** (QSemaphore *, const QString &, double *)
- void **getTextValue** (QSemaphore *, const QString &, QString *)

Public Member Functions

- void **zoom** (double)
- tc_items **allItems** ()
- tc_items **itemsOfFamily** (const char *)
- tc_items **itemsOfFamily** (const char *, tc_items)
- tc_items **selectedItems** ()
- long **find** (const char *)
- tc_items **findItems** (tc_strings)
- void **select** (long)
- void **deselect** ()
- const char * **getName** (long)
- const char * **getUniqueName** (long)
- void **setName** (long, const char *)
- tc_strings **getNames** (tc_items)
- tc_strings **getUniqueNames** (tc_items)
- const char * **getFamily** (long)
- int **isA** (long, const char *)
- void **removeItem** (long)
- void **setPos** (long, double, double)
- void **setPos** (tc_items, tc_matrix)
- tc_matrix **getPos** (tc_items)
- double **getY** (long)
- double **getX** (long)
- void **moveSelected** (double, double)
- void **outputTable** (tc_matrix m)
- void **outputText** (const char *)
- void **errorReport** (const char *)
- void **clearText** ()
- void **printFile** (const char *)
- void **createInputWindow** (tc_matrix, const char *, const char *)
- void **createInputWindow** (long, tc_matrix, const char *, MatrixInputFunction)
- void **createSliders** (long, tc_matrix, MatrixInputFunction)
- void **addInputWindowOptions** (const char *, int i, int j, tc_strings)
- void **addInputWindowCheckbox** (const char *, int i, int j)
- void **openNewWindow** (const char *)
- int **isWindows** ()
- int **isMac** ()
- int **isLinux** ()
- const char * **appDir** ()
- const char * **homeDir** ()
- tc_strings **getNumericalDataNames** (long)
- tc_strings **getTextDataNames** (long)
- tc_matrix **getNumericalData** (long, const char *)
- void **setNumericalData** (long, const char *, tc_matrix)

- tc_table **getTextData** (long, const char *)
- void **setTextData** (long, const char *, tc_table)
- tc_items **getChildren** (long)
- long **getParent** (long)
- const char * **getString** (const char *)
- const char * **getFilename** ()
- int **getSelectedString** (const char *, tc_strings, const char *)
- double **getNumber** (const char *)
- void **getNumbers** (tc_strings, double *)
- int **askQuestion** (const char *)
- void **messageDialog** (const char *)
- void **openFile** (const char *)
- void **saveToFile** (const char *)
- void **setSize** (long, double, double, int)
- double **getWidth** (long)
- double **getHeight** (long)
- void **setAngle** (long, double, int)
- const char * **getColor** (long)
- void **setColor** (long, const char *, int)
- void **changeGraphics** (long, const char *)
- void **changeArrowHead** (long, const char *)
- void **screenshot** (const char *, int, int)
- int **screenHeight** ()
- int **screenWidth** ()
- int **screenX** ()
- int **screenY** ()
- const char * **annotation** ()
- void **insertAnnotation** (const char *, double, double)
- void **setNumericalValues** (tc_matrix)
- void **setNumericalValue** (const char *, double)
- void **setTextValues** (tc_table)
- void **setTextValue** (const char *, const char *)
- double **getNumericalValue** (const char *)
- const char * **getTextValue** (const char *)

6.24.1 Detailed Description

Function to Signal converter for [MainWindow](#).

The documentation for this class was generated from the following files:

- C_API_Slots.h
- C_API_Slots.cpp

6.25 Tinkercell::CThread Class Reference

This class is used to run specific functions inside a C dynamic library as a separate thread. The class can be used to load a library or just run a specific function inside an already loaded library. If the library is loaded by this class, the library will be unloaded upon completion on the function. To prevent the automatic unloading, use the setAutoUnload option. Only four types of functions are supported.

```
#include <CThread.h>
```

Public Slots

- virtual void [unload](#) ()
unload the C library
- virtual void [update](#) ()
call the callback function, if one exists

Signals

- void [setProgress](#) (int)
display progress of this thread (0-100). This signal is usually connected to a slot in ProgressBarSignalItem
- void [setTitle](#) (const QString &)
- void [hideProgressBar](#) ()
hide the progress bar
- void [showProgressBar](#) ()
show the progress bar

Public Member Functions

- virtual void [showProgress](#) (const QString &, int)
show progress in a progress dialog
- [CThread](#) ([MainWindow](#) *main, QLibrary *lib=0, bool autoUnload=false)
constructor
- [CThread](#) ([MainWindow](#) *main, const QString &lib, bool autoUnload=false)
constructor
- virtual [~CThread](#) ()
destructor: unload and deletes the library
- virtual void [setFunction](#) (void(*f)(void), QSemaphore *sem=0)
set the function to run inside this threads
- virtual void [setFunction](#) (void(*f)(double), QSemaphore *sem=0)
set the function to run inside this threads
- virtual void [setFunction](#) (void(*f)(const char *), QSemaphore *sem=0)
set the function to run inside this threads
- virtual void [setFunction](#) (void(*f)(tc_matrix), QSemaphore *sem=0)
set the function to run inside this threads
- virtual void [setVoidFunction](#) (const char *, QSemaphore *sem=0)

set the function to run inside this threads

- virtual void [setDoubleFunction](#) (const char *, QSemaphore *sem=0)
set the function to run inside this threads
- virtual void [setCharFunction](#) (const char *, QSemaphore *sem=0)
set the function to run inside this threads
- virtual void [setMatrixFunction](#) (const char *, QSemaphore *sem=0)
set the function to run inside this threads
- virtual void [setLibrary](#) (QLibrary *)
set the dynamic library for this threads. The library will be loaded if it has not already been loaded
- virtual void [setLibrary](#) (const QString &)
set the dynamic library for this threads.
- virtual QLibrary * [library](#) ()
the library used inside this thread
- virtual void [setAutoUnload](#) (bool)
set whether or not to automatically unload the library when the thread is done running
- virtual bool [autoUnload](#) ()
whether or not to automatically unload the library when the thread is done running
- virtual void [setArg](#) (double, QSemaphore *sem=0)
set the argument for the target function
- virtual void [setArg](#) (const QString &, QSemaphore *sem=0)
set the argument for the target function
- virtual void [setArg](#) (const [DataTable](#)< qreal > &, QSemaphore *sem=0)
set the argument for the target function

Static Public Member Functions

- static QLibrary * [loadLibrary](#) (const QString &name, QObject *parent=0)
search the default tinkercell folders for the library and load it

Public Attributes

- [MainWindow](#) * [mainWindow](#)
main window

Static Public Attributes

- static QList< CThread * > [cthreads](#)
list stores pointers to different threads

Protected Slots

- virtual void [cleanupAfterTerminated](#) ()
cleanup (such as unload libraries) upon termination

Protected Member Functions

- virtual void [setupCFunctionPointers](#) (QLibrary *lib=0)
setup the C pointers in TC_Main.h
- virtual void [call_tc_main](#) ()
call tc_main
- virtual void [run](#) ()
the main function that runs one of the specified functions

Protected Attributes

- bool **hasDialog**
- bool [autoUnloadLibrary](#)
whether or not to automatically unload the library when the thread is done running
- void(* [f1](#))(void)
one of the functions that can be run inside this thread
- void(* [f2](#))(double)
one of the functions that can be run inside this thread
- void(* [f3](#))(const char *)
one of the functions that can be run inside this thread
- void(* [f4](#))(tc_matrix)
one of the functions that can be run inside this thread
- void(* [callbackPtr](#))(void)
callback function
- void(* [callWhenExitPtr](#))(void)
call when exit function
- QLibrary * [lib](#)

the library where the functions are located that can be run inside this thread

- double [argDouble](#)
the argument for one of the the run function
- QString [argString](#)
the argument for one of the the run function
- [DataTable](#)< qreal > [argMatrix](#)
the argument for one of the the run function

6.25.1 Detailed Description

This class is used to run specific functions inside a C dynamic library as a separate thread. The class can be used to load a library or just run a specific function inside an already loaded library. If the library is loaded by this class, the library will be unloaded upon completion on the function. To prevent the automatic unloading, use the setAutoUnload option. Only four types of functions are supported.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 `Tinkercell::CThread::CThread (MainWindow * main, QLibrary * lib = 0, bool autoUnload = false)`

constructor

Parameters

[MainWindow](#) the Tinkercell main window
[QLibrary](#) the dynamic library to load (optional)
bool whether or not to automatically unload the library

6.25.2.2 `Tinkercell::CThread::CThread (MainWindow * main, const QString & lib, bool autoUnload = false)`

constructor

Parameters

[MainWindow](#) the Tinkercell main window
[QString](#) the name of the dynamic library to load (optional)
bool whether or not to automatically unload the library

6.25.3 Member Function Documentation

6.25.3.1 `bool Tinkercell::CThread::autoUnload () [virtual]`

whether or not to automatically unload the library when the thread is done running

Returns

bool

6.25.3.2 QLibrary * TinkerCell::CThread::library () [virtual]

the library used inside this thread

Returns

QLibrary*

6.25.3.3 QLibrary * TinkerCell::CThread::loadLibrary (const QString & name, QObject * parent = 0) [static]

search the default tinkercell folders for the library and load it

Parameters

QString name of library (with or without full path)

QObject parent

Returns

QLibrary* the loaded library. 0 if cannot be loaded.

6.25.3.4 void TinkerCell::CThread::setArg (double d, QSemaphore * sem = 0) [virtual]

set the argument for the target function

Parameters

double

6.25.3.5 void TinkerCell::CThread::setArg (const QString & s, QSemaphore * sem = 0) [virtual]

set the argument for the target function

Parameters

QString

6.25.3.6 void TinkerCell::CThread::setArg (const DataTable< qreal > & dat, QSemaphore * sem = 0) [virtual]

set the argument for the target function

Parameters

DataTable

6.25.3.7 void Tinkercell::CThread::setAutoUnload (bool *b*) [virtual]

set whether or not to automatically unload the library when the thread is done running

Parameters

bool

6.25.3.8 void Tinkercell::CThread::setCharFunction (const char * *f*, QSemaphore * *sem* = 0) [virtual]

set the function to run inside this threads

Parameters

void name of the function inside the library that has been loaded in this thread.

6.25.3.9 void Tinkercell::CThread::setDoubleFunction (const char * *f*, QSemaphore * *sem* = 0) [virtual]

set the function to run inside this threads

Parameters

void name of the function inside the library that has been loaded in this thread.

6.25.3.10 void Tinkercell::CThread::setFunction (void(*) (void) *f*, QSemaphore * *sem* = 0) [virtual]

set the function to run inside this threads

Parameters

void function pointer

6.25.3.11 void Tinkercell::CThread::setFunction (void(*) (double) *f*, QSemaphore * *sem* = 0) [virtual]

set the function to run inside this threads

Parameters

void function pointer

6.25.3.12 void Tinkercell::CThread::setFunction (void(*) (const char *) *f*, QSemaphore * *sem* = 0) [virtual]

set the function to run inside this threads

Parameters

void function pointer

6.25.3.13 void Tinkercell::CThread::setFunction (void(*) (tc_matrix) *f*, QSemaphore * *sem* = 0) [virtual]

set the function to run inside this threads

Parameters

void function pointer

6.25.3.14 void Tinkercell::CThread::setLibrary (QLibrary * *lib*) [virtual]

set the dynamic library for this threads. The library will be loaded if it has not already been loaded

Parameters

*QLibrary** library

6.25.3.15 void Tinkercell::CThread::setLibrary (const QString & *libname*) [virtual]

set the dynamic library for this threads.

Parameters

*QLibrary** library

6.25.3.16 void Tinkercell::CThread::setMatrixFunction (const char * *f*, QSemaphore * *sem* = 0) [virtual]

set the function to run inside this threads

Parameters

void name of the function inside the library that has been loaded in this thread.

6.25.3.17 void Tinkercell::CThread::setTitle (const QString &) [signal]

set title of the dialog for this thread that shows the progress bar and kill button

6.25.3.18 void Tinkercell::CThread::setVoidFunction (const char * *f*, QSemaphore * *sem* = 0) [virtual]

set the function to run inside this threads

Parameters

void name of the function inside the library that has been loaded in this thread.

6.25.3.19 void Tinkercell::CThread::showProgress (const QString & *title*, int *progress*) [virtual]

show progress in a progress dialog

Parameters

QString title of the progress bar

int progress in range 0-100

The documentation for this class was generated from the following files:

- CThread.h
- CThread.cpp

6.26 Tinkercell::ConnectionGraphicsItem::CurveSegment Class Reference

A set of control points and two arrow heads.

```
#include <ConnectionGraphicsItem.h>
```

Public Member Functions

- **CurveSegment** (int)
- **CurveSegment** (int, [ConnectionGraphicsItem::ControlPoint](#) *)
- **CurveSegment** (const [CurveSegment](#) &)

Public Attributes

- [ArrowHeadItem](#) * **arrowStart**
- [ArrowHeadItem](#) * **arrowEnd**

6.26.1 Detailed Description

A set of control points and two arrow heads.

The documentation for this class was generated from the following files:

- ConnectionGraphicsItem.h
- ConnectionGraphicsItem.cpp

6.27 Tinkercell::DataTable< T > Class Template Reference

[DataTable](#) is a 2D vector with row names and column names.

```
#include <DataTable.h>
```

Public Member Functions

- virtual QString [description](#) () const
get description of this table
- virtual QString & [description](#) ()
get or set description of this table
- virtual QStringList [columnNames](#) () const
get the column names
- virtual bool [hasRow](#) (const QString &) const
check is this table has a row with the given name
- virtual bool [hasColumn](#) (const QString &) const
check is this table has a column with the given name
- virtual QStringList [rowNames](#) () const
get the row names
- virtual QString [rowName](#) (int i) const
get the ith row name reference. can be used to change the row name
- virtual QString [columnName](#) (int i) const
get the ith column name. cannot be used to change the column name
- virtual void [setRowName](#) (int i, const QString &name)
get the ith row name. cannot be used to change the row name
- virtual void [setColumnName](#) (int i, const QString &name)
get the ith column name reference. can be used to change the column name
- virtual void [setColumnNames](#) (const QStringList &names)
set all the column names.
- virtual void [setRowNames](#) (const QStringList &names)
set all the row names.
- virtual int [rows](#) () const
get the number of rows
- virtual int [columns](#) () const
get the number of columns
- virtual T & [value](#) (int i, int j=0)
get the value at the ith row and jth column. can also be used to set the value
- virtual T & [operator\(\)](#) (int i, int j=0)
get the value at the ith row and jth column. can also be used to set the value

- virtual T [operator\(\)](#) (int i, int j=0) const
get the value at the ith row and jth column. can also be used to set the value
- virtual T & [value](#) (const QString &r, const QString &c)
get the value using row and column names. can also be used to set the value. Fast lookup using hashtables.
- virtual T & [operator\(\)](#) (const QString &r, const QString &c)
get the value using row and column names. can also be used to set the value. Fast lookup using hashtables.
- virtual T [operator\(\)](#) (const QString &r, const QString &c) const
get the value using row and column names. can also be used to set the value. Fast lookup using hashtables.
- virtual T & [value](#) (const QString &r, int j=0)
get the value using row name. can also be used to set the value. Fast lookup using hashtables.
- virtual T & [operator\(\)](#) (const QString &r, int j=0)
get the value using row name and column index. can also be used to set the value. Fast lookup using hashtables.
- virtual T [operator\(\)](#) (const QString &r, int j=0) const
get the value using row name and column index. can also be used to set the value. Fast lookup using hashtables.
- virtual T & [value](#) (int i, const QString &c)
get the value using column name. can also be used to set the value. Fast lookup using hashtables.
- virtual T & [operator\(\)](#) (int i, const QString &c)
get the value using row name and column index. can also be used to set the value. Fast lookup using hashtables.
- virtual T [operator\(\)](#) (int i, const QString &c) const
get the value using row index and column name. can also be used to set the value. Fast lookup using hashtables.
- virtual bool [operator==](#) (const [DataTable](#)< T > &D)
checks if the two data table's headers and contents are the same
- virtual bool [operator!=](#) (const [DataTable](#)< T > &D)
exactly opposite of operator ==
- virtual T [at](#) (int i, int j=0) const
get the value using row and column number. cannot also be used to set the value.
- virtual T [at](#) (const QString &r, const QString &c) const
get the value using row and column name. cannot also be used to set the value.
- virtual T [at](#) (const QString &r, int j=0) const
get the value using row name. cannot also be used to set the value.
- virtual T [at](#) (int i, const QString &c) const

get the value using column name. cannot also be used to set the value.

- virtual void [resize](#) (int m, int n=1)
set the size of the data table
- virtual bool [insertRow](#) (int k, const QString &row)
insert a new row at the given location with the given name. Insertion will fail if there is already a row with the same name
- virtual bool [insertColumn](#) (int k, const QString &col)
insert a new column at the given location with the given name. Insertion will fail if there is already a column with the same name
- virtual bool [removeRow](#) (int k)
remove an existing row at the given index.
- virtual bool [removeRow](#) (const QString &name)
remove an existing row with the given name.
- virtual bool [removeColumn](#) (int k)
remove an existing column at the given index.
- virtual bool [removeColumn](#) (const QString &name)
remove an existing col with the given name.
- virtual void [swapRows](#) (int i1, int i2)
swap two rows. Nothing will happen if the given numbers are outside the table
- virtual void [swapColumns](#) (int j1, int j2)
swap two columns. Nothing will happen if the given numbers are outside the table
- virtual void [swapRows](#) (const QString &s1, const QString &s2)
swap two rows using their name. Nothing will happen if the given numbers are outside the table
- virtual void [swapColumns](#) (const QString &s1, const QString &s2)
swap two columns using their name. Nothing will happen if the given numbers are outside the table
- virtual [DataTable](#)< T > [transpose](#) () const
*get transpose of the table. complexity = $n*m$ (use sparingly)*

Protected Attributes

- QVector< T > [dataMatrix](#)
the values in the table
- QVector< QString > [colHeaders](#)
the column and row names
- QVector< QString > [rowHeaders](#)

- `QHash< QString, int >` [colHash](#)
hash for quick lookup of row and columns by name
- `QHash< QString, int >` **rowHash**
- `QString` [desc](#)
a description of this table (optional)

6.27.1 Detailed Description

`template<typename T> class Tinkercell::DataTable< T >`

[DataTable](#) is a 2D vector with row names and column names.

6.27.2 Member Function Documentation

6.27.2.1 `template<typename T> T Tinkercell::DataTable< T >::at (int i, int j = 0) const`
`[virtual]`

get the value using row and column number. cannot also be used to set the value.

Parameters

- int* row number
- int* column number (defaults to 0)

Returns

T copy of value at given row and column. returns value at 0 if row and column are not in the table

Parameters

- int* row number
- int* column number

Returns

T copy of value at given row and column. returns value at 0 if row and column are not in the table

6.27.2.2 `template<typename T> T Tinkercell::DataTable< T >::at (int i, const QString & c)`
`const [virtual]`

get the value using column name. cannot also be used to set the value.

Parameters

- int* row number
- int* column name

Returns

T copy of value at given row and column. returns value at 0 if row and column are not in the table

6.27.2.3 template<typename T > T TinkerCell::DataTable< T >::at (const QString & *r*, const QString & *c*) const [virtual]

get the value using row and column name. cannot also be used to set the value.

Parameters

QString row name

QString column name

Returns

T copy of value at given row and column. returns value at 0 if row and column are not in the table

6.27.2.4 template<typename T > T TinkerCell::DataTable< T >::at (const QString & *r*, int *j* = 0) const [virtual]

get the value using row name. cannot also be used to set the value.

Parameters

QString row name

int column number (defaults to 0)

Returns

T copy of value at given row and column. returns value at 0 if row and column are not in the table

Parameters

QString row name

int column number

Returns

T copy of value at given row and column. returns value at 0 if row and column are not in the table

6.27.2.5 template<typename T > QString TinkerCell::DataTable< T >::columnName (int *i*) const [virtual]

get the ith column name. cannot be used to change the column name

Parameters

int col number

Returns

QString copy of the ith column name

6.27.2.6 `template<typename T> QStringList TinkerCell::DataTable< T >::columnNames ()
const [virtual]`

get the column names

Returns

QStringList column names (copy)
QVector reference to the actual column names

6.27.2.7 `template<typename T> int TinkerCell::DataTable< T >::columns () const
[virtual]`

get the number of columns

Returns

int number of columns

6.27.2.8 `template<typename T> bool TinkerCell::DataTable< T >::hasColumn (const QString
& s) const [virtual]`

check if this table has a column with the given name

Parameters

QString column name

Returns

bool true if the column with the name exists

6.27.2.9 `template<typename T> bool TinkerCell::DataTable< T >::hasRow (const QString & s
) const [virtual]`

check if this table has a row with the given name

Parameters

QString row name

Returns

bool true if the row with the name exists

6.27.2.10 `template<typename T> bool TinkerCell::DataTable< T >::insertColumn (int k,
const QString & col) [virtual]`

insert a new column at the given location with the given name. Insertion will fail if there is already a column with the same name

Parameters

int column number
QString column name

Returns

Boolean false if failed, true if successful

6.27.2.11 `template<typename T> bool Tinkercell::DataTable< T >::insertRow (int k, const QString & row) [virtual]`

insert a new row at the given location with the given name. Insertion will fail if there is already a row with the same name

Parameters

int row number
QString row name

Returns

Boolean false if failed, true if successful

6.27.2.12 `template<typename T> bool Tinkercell::DataTable< T >::operator!= (const DataTable< T > & D) [virtual]`

exactly opposite of operator ==

Parameters

DataTable<T>

Returns

bool

6.27.2.13 `template<typename T> T & Tinkercell::DataTable< T >::operator() (int i, int j = 0) [virtual]`

get the value at the ith row and jth column. can also be used to set the value

Parameters

int row number
int column number (defaults to 0)

Returns

T reference to value at ith row and jth column. returns value at 0 if i or j are not inside the table

6.27.2.14 `template<typename T> T Tinkercell::DataTable< T >::operator() (int i, int j = 0)
const [virtual]`

get the value at the ith row and jth column. can also be used to set the value

Parameters

int row number

int column number (defaults to 0)

Returns

T value at ith row and jth column. returns value at 0 if i or j are not inside the table

6.27.2.15 `template<typename T> T & Tinkercell::DataTable< T >::operator() (const QString
& r, const QString & c) [virtual]`

get the value using row and column names. can also be used to set the value. Fast lookup using hashtables.

Parameters

QString row name

QString column name

Returns

T reference to value at given row and column. returns value at 0 if row and column are not in the table

6.27.2.16 `template<typename T> T Tinkercell::DataTable< T >::operator() (const QString &
r, const QString & c) const [virtual]`

get the value using row and column names. can also be used to set the value. Fast lookup using hashtables.

Parameters

QString row name

QString column name

Returns

T value at given row and column. returns value at 0 if row and column are not in the table

6.27.2.17 `template<typename T> T & Tinkercell::DataTable< T >::operator() (const QString
& r, int j = 0) [virtual]`

get the value using row name and column index. can also be used to set the value. Fast lookup using hashtables.

Parameters

QString row name

QString column index

Returns

T reference to value at given row and column. returns value at 0 if row and column are not in the table

6.27.2.18 `template<typename T > T Tinkercell::DataTable< T >::operator() (const QString & r, int j = 0) const [virtual]`

get the value using row name and column index. can also be used to set the value. Fast lookup using hashtables.

Parameters

QString row name

QString column index

Returns

T value at given row and column. returns value at 0 if row and column are not in the table

6.27.2.19 `template<typename T > T & Tinkercell::DataTable< T >::operator() (int i, const QString & c) [virtual]`

get the value using row name and column index. can also be used to set the value. Fast lookup using hashtables.

Parameters

QString row index

QString column name

Returns

T reference to value at given row and column. returns value at 0 if row and column are not in the table

6.27.2.20 `template<typename T > T Tinkercell::DataTable< T >::operator() (int i, const QString & c) const [virtual]`

get the value using row index and column name. can also be used to set the value. Fast lookup using hashtables.

Parameters

QString row index

QString column name

Returns

T value at given row and column. returns value at 0 if row and column are not in the table

6.27.2.21 `template<typename T> bool Tinkercell::DataTable< T >::operator== (const DataTable< T > & D) [virtual]`

checks if the two data table's headers and contents are the same

Parameters

DataTable<T>

Returns

bool

6.27.2.22 `template<typename T> bool Tinkercell::DataTable< T >::removeColumn (int k) [virtual]`

remove an existing column at the given index.

Parameters

int column number

Returns

Boolean false if failed, true if successful

6.27.2.23 `template<typename T> bool Tinkercell::DataTable< T >::removeColumn (const QString & name) [virtual]`

remove an existing col with the given name.

Parameters

QString row name

Returns

Boolean false if failed, true if successful

6.27.2.24 `template<typename T> bool Tinkercell::DataTable< T >::removeRow (int k) [virtual]`

remove an existing row at the given index.

Parameters

int row number

Returns

Boolean false if failed, true if successful

6.27.2.25 `template<typename T> bool TinkerCell::DataTable< T >::removeRow (const QString & name) [virtual]`

remove an existing row with the given name.

Parameters

QString row name

Returns

Boolean false if failed, true if successful

6.27.2.26 `template<typename T> void TinkerCell::DataTable< T >::resize (int m, int n = 1) [virtual]`

set the size of the data table

Parameters

int row count

int column count (defaults to 1)

Returns

void

Parameters

int row count

int column count

Returns

void

6.27.2.27 `template<typename T> QString TinkerCell::DataTable< T >::rowName (int i) const [virtual]`

get the ith row name reference. can be used to change the row name

Parameters

int col number

Returns

QString copy to the ith row name

6.27.2.28 `template<typename T> QStringList TinkerCell::DataTable< T>::rowNames ()
const [virtual]`

get the row names

Returns

QStringList row names (copy)
QVector reference to the actual row names

6.27.2.29 `template<typename T> int TinkerCell::DataTable< T>::rows () const [virtual]`

get the number of rows

Returns

int number of rows

6.27.2.30 `template<typename T> void TinkerCell::DataTable< T>::setColumnName (int i,
const QString & name) [virtual]`

get the ith column name reference. can be used to change the column name

Parameters

int col number
QString name

Returns

QString reference to the ith column name

6.27.2.31 `template<typename T> void TinkerCell::DataTable< T>::setColumnNames (const
QStringList & lst) [virtual]`

set all the column names.

Parameters

QStringList vector of strings

Returns

void

6.27.2.32 `template<typename T> void TinkerCell::DataTable< T>::setRowName (int i, const
QString & name) [virtual]`

get the ith row name. cannot be used to change the row name

Parameters

int row number

QString name

Returns

QString reference of the ith row name

Parameters

int row number

Returns

QString reference of the ith row name

6.27.2.33 `template<typename T> void TinkerCell::DataTable< T >::setRowNames (const QStringList & lst) [virtual]`

set all the row names.

Parameters

QStringList vector of strings

Returns

void

6.27.2.34 `template<typename T> void TinkerCell::DataTable< T >::swapColumns (int j1, int j2) [virtual]`

swap two columns. Nothing will happen if the given numbers are outside the table

Parameters

int first column number

int second column number

Returns

void

6.27.2.35 `template<typename T> void TinkerCell::DataTable< T >::swapColumns (const QString & s1, const QString & s2) [virtual]`

swap two columns using their name. Nothing will happen if the given numbers are outside the table

Parameters

int first column name

int second column name

Returns

void

6.27.2.36 `template<typename T> void Tinkercell::DataTable< T>::swapRows (int i1, int i2) [virtual]`

swap two rows. Nothing will happen if the given numbers are outside the table

Parameters

int first row number
int second row number

Returns

void

6.27.2.37 `template<typename T> void Tinkercell::DataTable< T>::swapRows (const QString & s1, const QString & s2) [virtual]`

swap two rows using their name. Nothing will happen if the given numbers are outside the table

Parameters

int first row name
int second row name

Returns

void

6.27.2.38 `template<typename T> DataTable< T> Tinkercell::DataTable< T>::transpose () const [virtual]`

get transpose of the table. complexity = n*m (use sparingly)

Returns

DataTable<T> new data table
 new data table

6.27.2.39 `template<typename T> T & Tinkercell::DataTable< T>::value (const QString & r, int j = 0) [virtual]`

get the value using row name. can also be used to set the value. Fast lookup using hashtables.

get the value using row name. can also be used to set the value. Slower than using [value\(int,int\)](#)

Parameters

QString row name
int column number (defaults to 0)

Returns

T reference to value at given row and column. returns value at 0 if row and column are not in the table

Parameters

QString row name

int column number

Returns

T reference to value at given row and column. returns value at 0 if row and column are not in the table

6.27.2.40 `template<typename T > T & Tinkercell::DataTable< T >::value (int i, int j = 0)
[virtual]`

get the value at the ith row and jth column. can also be used to set the value

Parameters

int row number

int column number (defaults to 0)

Returns

T reference to value at ith row and jth column. returns value at 0 if i or j are not inside the table

Parameters

int row number (i)

int column number (j)

Returns

T reference to value at ith row and jth column. returns value at 0 if i or j are not inside the table

6.27.2.41 `template<typename T > T & Tinkercell::DataTable< T >::value (const QString & r,
const QString & c) [virtual]`

get the value using row and column names. can also be used to set the value. Fast lookup using hashtables.

get the value using row and column names. can also be used to set the value. Slower than using [value\(int,int\)](#)

Parameters

QString row name

QString column name

Returns

T reference to value at given row and column. returns value at 0 if row and column are not in the table

6.27.2.42 `template<typename T> T & TinkerCell::DataTable< T >::value (int i, const QString & c) [virtual]`

get the value using column name. can also be used to set the value. Fast lookup using hashtables.

get the value using column name. can also be used to set the value. Slower than using [value\(int,int\)](#)

Parameters

int row number

QString column name

Returns

T reference to value at given row and column. returns value at 0 if row and column are not in the table

The documentation for this class was generated from the following file:

- `DataTable.h`

6.28 TinkerCell::GraphicsScene Class Reference

The primary task of the graphics scene is to draw items. All interactions with the [GraphicsScene](#) is done through [MainWindow](#) or [NetworkHandle](#). [NetworkHandle](#) provides functions such as move, insert, and remove. [MainWindow](#) relays all the signals, such as mouse and key events, from the [GraphicsScene](#). So, there is rarely a need to directly interact with the [GraphicsScene](#).

```
#include <GraphicsScene.h>
```

Public Slots

- virtual void [fitAll](#) () const
adjusts view to include all items
- virtual void [fitInView](#) (const QRectF &) const
adjusts view to include the given rect
- virtual QRect [mapToWidget](#) (QRectF rect=QRectF(0, 0, 0, 0)) const
map a rect from the scene coordinates to the view coordinates
- virtual void [popOut](#) ()
calls main window's popOut
- virtual void [popIn](#) ()
calls main window's popIn
- virtual void [zoom](#) (qreal scaleFactor)
zoom in or out
- virtual void [zoomIn](#) ()
zoom in (zoom with 1.5)

- virtual void [zoomOut](#) ()
zoom out (zoom with 0.75)
- virtual void [selectAll](#) ()
select all items
- virtual void [find](#) (const QString &, bool clearSelected=true)
select items with the given text
- virtual void [find](#) (const QStringList &)
select items with the given texts
- virtual void [deselect](#) ()
deselect all selected items
- virtual void [copy](#) ()
copy selected items
- virtual void [cut](#) ()
cut selected items
- virtual void [paste](#) ()
paste copied items
- virtual void [move](#) (QGraphicsItem *item, const QPointF &distance)
a simple move operation that also adds undo command to history window and emits associated signal(s)
- virtual void [move](#) (const QList< QGraphicsItem * > &items, const QPointF &distance)
a simple move operation that also adds undo command to history window and emits associated signal(s)
- virtual void [move](#) (const QList< QGraphicsItem * > &items, const QList< QPointF > &distance)
a simple move operation that also adds undo command to history window and emits associated signal(s)
- virtual void [insert](#) (const QString &name, QGraphicsItem *item)
this command performs an insert and also adds undo command to history window and emits associated signal(s)
- virtual void [insert](#) (const QString &name, const QList< QGraphicsItem * > &items)
this command performs an insert and also adds undo command to history window and emits associated signal(s)
- virtual void [remove](#) (const QString &name, QGraphicsItem *item)
this command performs an removal and also adds undo command to history window and emits associated signal(s)
- virtual void [remove](#) (const QString &name, const QList< QGraphicsItem * > &items)
this command performs an removal and also adds undo command to history window and emits associated signal(s)

- virtual void [removeSelected](#) ()
remove selected items
- virtual void [setBrush](#) (const QString &name, QGraphicsItem *item, const QBrush &to)
this command changes the brush of an item
- virtual void [setBrush](#) (const QString &name, const QList< QGraphicsItem * > &items, const QList< QBrush > &to)
this command changes the brush of an item and also adds undo command to history window and emits associated signal(s)
- virtual void [setZValue](#) (const QString &name, QGraphicsItem *item, qreal to)
this command changes the z value of an item and also adds undo command to history window and emits associated signal(s)
- virtual void [setZValue](#) (const QString &name, const QList< QGraphicsItem * > &items, const QList< qreal > &to)
this command changes the z value of an item and also adds undo command to history window and emits associated signal(s)
- virtual void [setPen](#) (const QString &name, QGraphicsItem *item, const QPen &to)
this command changes the pen of an item and also adds undo command to history window and emits associated signal(s)
- virtual void [setPen](#) (const QString &name, const QList< QGraphicsItem * > &items, const QList< QPen > &to)
this command changes the pen of an item and also adds undo command to history window and emits associated signal(s)
- virtual void [setBrushAndPen](#) (const QString &name, QGraphicsItem *item, const QBrush &brush, const QPen &pen)
this command changes the pen and/or brush of an item and also adds undo command to history window and emits associated signal(s)
- virtual void [setBrushAndPen](#) (const QString &name, const QList< QGraphicsItem * > &items, const QList< QBrush > &brushes, const QList< QPen > &pens)
this command changes the pen and/or brush of an item and also adds undo command to history window and emits associated signal(s)
- virtual void [transform](#) (const QString &name, QGraphicsItem *item, const QPointF &sizechange, qreal anglechange=0.0, bool VFlip=false, bool HFlip=false)
this command changes the size, angle, and orientation of an item and also adds undo command to history window and emits associated signal(s)
- virtual void [transform](#) (const QString &name, const QList< QGraphicsItem * > &items, const QList< QPointF > &sizechange, const QList< qreal > &anglechange=QList< qreal >(), const QList< bool > &verticalFlip=QList< bool >(), const QList< bool > &horizontalFlip=QList< bool >())
this command changes the size, angle, and orientation of an item and also adds undo command to history window and emits associated signal(s)

- virtual void [setParentItem](#) (const QString &name, QGraphicsItem *item, QGraphicsItem *newParent)
this command changes the parent of an item and also adds undo command to history window and emits associated signal(s)
- virtual void [setParentItem](#) (const QString &name, const QList< QGraphicsItem * > &items, QGraphicsItem *newParent)
this command changes the parent of an item and also adds undo command to history window and emits associated signal(s)
- virtual void [setParentItem](#) (const QString &name, const QList< QGraphicsItem * > &items, const QList< QGraphicsItem * > &newParents)
this command changes the parent of an item and also adds undo command to history window and emits associated signal(s)

Signals

- void [copyItems](#) (GraphicsScene *scene, QList< QGraphicsItem * > &, QList< [ItemHandle](#) * > &)
signals just before items are copied
- void [itemsAboutToBeRemoved](#) (GraphicsScene *scene, QList< QGraphicsItem * > &, QList< [ItemHandle](#) * > &, QList< [QUndoCommand](#) * > &)
signals just before items are deleted
- void [itemsRemoved](#) (GraphicsScene *scene, const QList< QGraphicsItem * > &, const QList< [ItemHandle](#) * > &)
signals whenever items are deleted
- void [itemsAboutToBeInserted](#) (GraphicsScene *scene, QList< QGraphicsItem * > &, QList< [ItemHandle](#) * > &, QList< [QUndoCommand](#) * > &)
signals whenever items are going to be added
- void [itemsInserted](#) (GraphicsScene *scene, const QList< QGraphicsItem * > &, const QList< [ItemHandle](#) * > &)
signals whenever items are added
- void [itemsSelected](#) (GraphicsScene *scene, const QList< QGraphicsItem * > &items, QPointF point, Qt::KeyboardModifiers modifiers)
signals whenever items are selected (item can be sub-item, not top-level)
- void [mousePressed](#) (GraphicsScene *scene, QPointF point, Qt::MouseButton, Qt::KeyboardModifiers modifiers)
signals whenever an empty node of the screen is clicked
- void [mouseReleased](#) (GraphicsScene *scene, QPointF point, Qt::MouseButton, Qt::KeyboardModifiers modifiers)
signals whenever an empty node of the screen is clicked

- void [mouseDoubleClicked](#) ([GraphicsScene](#) *scene, QPointF point, QGraphicsItem *, Qt::MouseButton, Qt::KeyboardModifiers modifiers)
emits event when mouse is double clicked
- void [mouseDragged](#) ([GraphicsScene](#) *scene, QPointF from, QPointF to, Qt::MouseButton, Qt::KeyboardModifiers modifiers)
signals whenever mouse is dragged from one point to another
- void [itemsAboutToBeMoved](#) ([GraphicsScene](#) *scene, QList< QGraphicsItem * > &item, QList< QPointF > &distance, QList< [QUndoCommand](#) * > &)
signals whenever items are going to be moved (each item is the top-most item)
- void [itemsMoved](#) ([GraphicsScene](#) *scene, const QList< QGraphicsItem * > &item, const QList< QPointF > &distance)
signals whenever items are being moved (each item is the top-most item)
- void [mouseMoved](#) ([GraphicsScene](#) *scene, QGraphicsItem *item, QPointF point, Qt::MouseButton, Qt::KeyboardModifiers modifiers, QList< QGraphicsItem * > &)
signals whenever mouse moves, and indicates whether it is on top of an item
- void [mouseOnTopOf](#) ([GraphicsScene](#) *scene, QGraphicsItem *item, QPointF point, Qt::KeyboardModifiers modifiers, QList< QGraphicsItem * > &)
signals whenever mouse is on top of an item
- void [sceneRightClick](#) ([GraphicsScene](#) *scene, QGraphicsItem *item, QPointF point, Qt::KeyboardModifiers modifiers)
signals whenever right click is made on an item or scene
- void [keyPressed](#) ([GraphicsScene](#) *scene, QKeyEvent *)
signals whenever a key is pressed
- void [keyReleased](#) ([GraphicsScene](#) *scene, QKeyEvent *)
signals whenever a key is released
- void [escapeSignal](#) (const QWidget *sender)
signals whenever the current activities need to be stopped
- void [filesDropped](#) (const QList< QFileInfo > &files)
signals whenever file(s) are dropped on the canvas
- void [colorChanged](#) ([GraphicsScene](#) *scene, const QList< QGraphicsItem * > &items)
signals whenever color of items are changed
- void [parentItemChanged](#) ([GraphicsScene](#) *scene, const QList< QGraphicsItem * > &items, const QList< QGraphicsItem * > &parents)
signals whenever item parents are changed

Public Member Functions

- **MainWindow** * **mainWindow** () const
the main window for this network
- **ConsoleWindow** * **console** () const
same as network->mainWindow->console()
- **ItemHandle** * **localHandle** () const
same as networkWindow->handle
- **ItemHandle** * **globalHandle** () const
same as network->globalHandle()
- virtual QRectF **visibleRegion** () const
Returns the currently visible window from the current graphics view.
- virtual void **setBackground** (const QPixmap &) const
set the background image for the scene
- virtual void **setForeground** (const QPixmap &) const
set the foreground image for the scene
- virtual QPointF & **lastPoint** ()
Returns the point where mouse was clicked last on the scene coordinates.
- virtual QPoint & **lastScreenPoint** ()
Returns the point where mouse was clicked last on the screen coordinates.
- virtual QList< QGraphicsItem * > & **selected** ()
Returns the list of pointers to items that are currently selected.
- virtual QRectF **selectedRect** ()
Returns a rectangle that includes all the selected items.
- virtual QList< QGraphicsItem * > & **moving** ()
Returns the list of pointers to items that are currently being moved.
- virtual qreal **ZValue** ()
top Z value
- **GraphicsScene** (NetworkHandle *network)
Constructor: sets 10000x10000 scene.
- virtual ~**GraphicsScene** ()
destructor
- virtual void **enableGrid** (int sz=100)
set the grid mode ON with the given grid size

- virtual void [disableGrid](#) ()
set the grid mode OFF, which is same as setting grid size to 0
- virtual void [setGridSize](#) (int sz=100)
set the grid size. If > 0, grid will be enabled. If 0, grid will be disabled
- virtual int [gridSize](#) () const
get the grid size being used (0 = no grid)
- virtual void [addItem](#) (QGraphicsItem *item)
Add a new item to the scene (different from insert).
- virtual void [centerOn](#) (const QPointF &point) const
place center at the point
- virtual void [clearSelection](#) ()
Clear all selection and moving items list.
- virtual void [print](#) (QPaintDevice *printer, const QRectF &rect=QRectF())
send everything on the screen to a printer
- virtual void [select](#) (QGraphicsItem *item)
select one item (does not deselect other items)
- virtual void [select](#) (const QList< QGraphicsItem * > &item)
select items (does not deselect previously selected items)
- virtual void [deselect](#) (QGraphicsItem *item)
deselect one item
- virtual void [showToolTip](#) (QPointF, const QString &)
show a tooltip a the given position
- virtual void [snapToGrid](#) (QGraphicsItem *)
snap the node item to the grid

Public Attributes

- [NetworkHandle](#) * [network](#)
the network represented by this scene
- [NetworkWindow](#) * [networkWindow](#)
the network window widget inside of which this scene is located
- bool [useDefaultBehavior](#)
indicates whether this scene is free to perform actions
- QMenu * [contextItemsMenu](#)

the context menu that is shown during right-click event on selected graphical items. Plugins can add new actions to this menu.

- QMenu * [contextScreenMenu](#)

the context menu that is shown during right-click event on the scene. Plugins can add new actions to this menu.

Static Public Attributes

- static bool [USE_DEFAULT_BEHAVIOR](#) = true

each graphics scene has a default behavior, i.e. moving, selecting, deleting. Whether or not to use the default behavior is set using scene->useDefaultBehavior. This static variable is the default value for each scene's useDefaultBehavior variable, i.e. setting this to true will cause a newly constructed graphics scene to NOT use default behaviors.

- static int [GRID](#) = 0

setting grid to a non-zero value forces node items to "fit" on the grid, where the gap between the grid lines is determined by this variable. The default is 0, i.e. no grid

- static QPen [SelectionRectanglePen](#) = Qt::NoPen

pen that is used to draw the selection rectangle

- static QBrush [SelectionRectangleBrush](#) = QBrush(QColor(0,132,255,50))

brush that is used to color the selection rectangle

- static QBrush [BackgroundBrush](#) = Qt::NoBrush

brush used to draw the background for all scenes

- static QColor [BackgroundColor](#)

background color for all scenes

- static QPen [GridPen](#) = QPen(Qt::lightGray,2)

pen used to draw the grid for the scene

- static QBrush [ForegroundBrush](#) = Qt::NoBrush

brush used to draw the foreground for the scene

- static QBrush [ToolTipBackgroundBrush](#) = QBrush(QColor(36,28,28,125))

brush used to draw the background of tool tips

- static QBrush [ToolTipTextBrush](#) = QBrush(QColor(255,255,255,255))

brush used to draw the text for tool tips

- static qreal [MIN_DRAG_DISTANCE](#) = 2.0

the minimum distance that gets classified as a "drag". Anything less will be considered just a click.

Protected Member Functions

- virtual void [hideToolTips](#) ()
hide the all tool tips
- virtual void [hideGraphicalTools](#) ()
hide the all graphical tools
- virtual void [showGraphicalTools](#) ()
show graphical tools for selected items
- virtual void [scaleGraphicalTools](#) ()
scale the visible graphical tools according to viewport size
- virtual void [mousePressEvent](#) (QGraphicsSceneMouseEvent *mouseEvent)
when mouse is pressed, the item at the position is added to selected list and moving list
- virtual void [mouseDoubleClickEvent](#) (QGraphicsSceneMouseEvent *mouseEvent)
when mouse is double clicked, the item at the position is added to selected list and moving list
- virtual void [mouseMoveEvent](#) (QGraphicsSceneMouseEvent *mouseEvent)
when mouse is moving, all items in moving list are moved
- virtual void [mouseReleaseEvent](#) (QGraphicsSceneMouseEvent *mouseEvent)
when mouse is released, moving list is cleared
- virtual void [keyPressEvent](#) (QKeyEvent *event)
when key is pressed
- virtual void [keyReleaseEvent](#) (QKeyEvent *event)
when key is released
- virtual void [contextMenuEvent](#) (QGraphicsSceneContextMenuEvent *contextMenuEvent)
context menu for the scene
- virtual void [populateContextMenu](#) ()
populate the context menu using selected items' tools actions
- virtual void [drawBackground](#) (QPainter *painter, const QRectF &rect)
draw background grid if in grid mode
- virtual void [selectConnections](#) (const QPointF &)
used to select the entire connection during mouse click

Static Protected Member Functions

- static void [clearStaticItems](#) ()
clears copied items

Protected Attributes

- int [gridSz](#)
grid size. If zero, then disabled
- qreal [lastZ](#)
topmost Z value
- bool [contextMenuJustActivated](#)
a hack to prevent strange mouse movements after context menu event
- QGraphicsRectItem [selectionRect](#)
rectangular selection area
- QList< QGraphicsItem * > [toolTips](#)
list of temporary tool tips
- QPointF [clickedPoint](#)
point where mouse is clicked
- QPoint [clickedScreenPoint](#)
point where mouse is clicked on the screen
- Qt::MouseButton [clickedButton](#)
button that was used when mouse was clicked
- bool [mouseDown](#)
mouse is being pressed
- QList< QGraphicsItem * > [selectedItems](#)
list of pointers to selected items
- QList< ToolGraphicsItem * > [visibleTools](#)
list of pointers to tool items
- QList< QGraphicsItem * > [movingItems](#)
list of pointers to moving items
- QGraphicsItemGroup * [movingItemsGroup](#)
group of moving items

Static Protected Attributes

- static QList< QGraphicsItem * > [duplicateItems](#)
used to store copied items
- static [GraphicsScene](#) * [copiedFromScene](#)
used to store copied items

Friends

- class **MainWindow**
- class **NetworkWindow**
- class **NetworkHandle**
- class **GraphicsView**
- class **SymbolsTable**

6.28.1 Detailed Description

The primary task of the graphics scene is to draw items. All interactions with the [GraphicsScene](#) is done through [MainWindow](#) or [NetworkHandle](#). [NetworkHandle](#) provides functions such as move, insert, and remove. [MainWindow](#) relays all the signals, such as mouse and key events, from the [GraphicsScene](#). So, there is rarely a need to directly interact with the [GraphicsScene](#).

6.28.2 Member Function Documentation

6.28.2.1 void TinkerCell::GraphicsScene::addItem (QGraphicsItem * item) [virtual]

Add a new item to the scene (different from insert).

Add a new item to the scene Precondition: None Postcondition: None.

See also

[insert](#)

Parameters

*QGraphicsItem** TinkerCell object

Returns

void

Parameters

TinkerCell object

Returns

void

6.28.2.2 void TinkerCell::GraphicsScene::centerOn (const QPointF & point) const [virtual]

place center at the point

place center at the point Precondition: None Postcondition: None

Parameters

QPointF point

Returns

void

Parameters

point

Returns

void

6.28.2.3 void Tinkercell::GraphicsScene::clearSelection () [virtual]

Clear all selection and moving items list.

Clear all selection and moving items list Precondition: None Postcondition: None.

Returns

void

6.28.2.4 void Tinkercell::GraphicsScene::colorChanged (GraphicsScene * scene, const QList< QGraphicsItem * > & items) [signal]

signals whenever color of items are changed

Parameters

GraphicsScene * scene where the event took place

QList<QGraphicsItem>&* items that changed color

Returns

void

6.28.2.5 void Tinkercell::GraphicsScene::contextMenuEvent (QGraphicsSceneContextMenuEvent * mouseEvent) [protected, virtual]

context menu for the scene

context menu for the scene Precondition: None Postcondition: None

Parameters

QGraphicsSceneContextMenuEvent * context menu event

Returns

void

Parameters

context menu event

Returns

void

6.28.2.6 void TinkerCell::GraphicsScene::copyItems (GraphicsScene * *scene*, QList< QGraphicsItem * > &, QList< ItemHandle * > &) [signal]

signals just before items are copied

Parameters

GraphicsScene * scene where the items are going to be copied

QList<QGraphicsItem>&* list of graphics items going to be copied

QList<ItemHandle>&* list of handles going to be copied (does NOT have to be the same number as items removed)

Returns

void

6.28.2.7 void TinkerCell::GraphicsScene::deselect () [virtual, slot]

deselect all selected items

deselect items

Returns

void

6.28.2.8 void TinkerCell::GraphicsScene::deselect (QGraphicsItem * *item*) [virtual]

deselect one item

deselect items

Parameters

*QGraphicsItem** item to deselect

Returns

void

6.28.2.9 void TinkerCell::GraphicsScene::disableGrid () [virtual]

set the grid mode OFF, which is same as setting grid size to 0

Returns

void

6.28.2.10 void TinkerCell::GraphicsScene::enableGrid (int sz = 100) [virtual]

set the grid mode ON with the given grid size

Parameters

double grid size (0 will disable grid)

Returns

void

6.28.2.11 void TinkerCell::GraphicsScene::escapeSignal (const QWidget * sender) [signal]

signals whenever the current activities need to be stopped

Parameters

QWidget * the widget that send the signal

Returns

void

6.28.2.12 void TinkerCell::GraphicsScene::filesDropped (const QList< QFileInfo > & files) [signal]

signals whenever file(s) are dropped on the canvas

Parameters

QList< QFileInfo > & the name(s) of the file(s)

Returns

void

6.28.2.13 void TinkerCell::GraphicsScene::fitAll () const [virtual, slot]

adjusts view to include all items

Returns

void

6.28.2.14 void TinkerCell::GraphicsScene::fitInView (const QRectF & rect) const [virtual, slot]

adjusts view to include the given rect

adjusts view to include rect

Parameters*QRectF***Returns**

void

6.28.2.15 int Tinkercell::GraphicsScene::gridSize () const [virtual]

get the grid size being used (0 = no grid)

Returns

int

6.28.2.16 void Tinkercell::GraphicsScene::insert (const QString & name, const QList< QGraphicsItem * > & items) [virtual, slot]

this command performs an insert and also adds undo command to history window and emits associated signal(s)

this command performs an insert and allows redo/undo of that insert

6.28.2.17 void Tinkercell::GraphicsScene::insert (const QString & name, QGraphicsItem * item) [virtual, slot]

this command performs an insert and also adds undo command to history window and emits associated signal(s)

Parameters*QString* name of new item*QList<QPointF>&* distance to move the items specified for each item**Returns**

void

6.28.2.18 void Tinkercell::GraphicsScene::itemsAboutToBeInserted (GraphicsScene * scene, QList< QGraphicsItem * > &, QList< ItemHandle * > &, QList< QUndoCommand * > &) [signal]

signals whenever items are going to be added

Parameters*GraphicsScene** scene where the items are added*QList<QGraphicsItem*>&* list of new graphics items*QList<ItemHandle*>&* list of new handles (does NOT have to be the same number as items)

QList<QUndoCommand>&* list of commands that will be executed right before items are inserted

Returns

void

6.28.2.19 void TinkerCell::GraphicsScene::itemsAboutToBeMoved (GraphicsScene * *scene*,
QList< QGraphicsItem * > & *item*, QList< QPointF > & *distance*, QList<
QUndoCommand * > &) [signal]

signals whenever items are going to be moved (each item is the top-most item)

Parameters

*GraphicsScene** *scene* where the items were moved

QList<QGraphicsItem>&* list of pointers to all moving items

QPointF distance by which items moved

Qt::KeyboardModifiers modifier keys being used when mouse clicked

QList<QUndoCommand>&* list of commands that will be executed right before items are inserted

Returns

void

6.28.2.20 void TinkerCell::GraphicsScene::itemsAboutToBeRemoved (GraphicsScene * *scene*,
QList< QGraphicsItem * > &, QList< ItemHandle * > &, QList< QUndoCommand
* > &) [signal]

signals just before items are deleted

Parameters

GraphicsScene * *scene* where the items are going to be removed

QList<QGraphicsItem>&* list of graphics items going to be removed

QList<ItemHandle>&* list of handles going to be removed (does NOT have to be the same number as items removed)

QList<QUndoCommand>&* list of commands that will be executed right before items are removed

Returns

void

6.28.2.21 void TinkerCell::GraphicsScene::itemsInserted (GraphicsScene * *scene*, const QList<
QGraphicsItem * > &, const QList< ItemHandle * > &) [signal]

signals whenever items are added

Parameters

*GraphicsScene** *scene* where the items were added

QList<QGraphicsItem>&* list of new graphics items

QList<ItemHandle>&* list of new handles (does NOT have to be the same number as items)

Returns

void

6.28.2.22 void Tinkercell::GraphicsScene::itemsMoved (QGraphicsScene * *scene*, const QList< QGraphicsItem * > & *item*, const QList< QPointF > & *distance*) [signal]

signals whenever items are being moved (each item is the top-most item)

Parameters

*GraphicsScene** *scene* where the items were moved

QList<QGraphicsItem>&* list of pointers to all moving items

QPointF distance by which items moved

Qt::KeyboardModifiers modifier keys being used when mouse clicked

Returns

void

6.28.2.23 void Tinkercell::GraphicsScene::itemsRemoved (QGraphicsScene * *scene*, const QList< QGraphicsItem * > &, const QList< ItemHandle * > &) [signal]

signals whenever items are deleted

Parameters

*GraphicsScene** *scene* where the items were removed

QList<QGraphicsItem>&* list of items removed

QList<ItemHandle>&* list of handles removed (does NOT have to be the same number as items removed)

Returns

void

6.28.2.24 void Tinkercell::GraphicsScene::itemsSelected (QGraphicsScene * *scene*, const QList< QGraphicsItem * > & *items*, QPointF *point*, Qt::KeyboardModifiers *modifiers*) [signal]

signals whenever items are selected (item can be sub-item, not top-level)

Parameters

*GraphicsScene** *scene* where items are selected

QList<QGraphicsItem>&* list of all selected item pointers

QPointF point where mouse is clicked

Qt::KeyboardModifiers modifier keys being used when mouse clicked

Returns

void

6.28.2.25 void Tinkercell::GraphicsScene::keyPressed (GraphicsScene * *scene*, QKeyEvent *)
[signal]

signals whenever a key is pressed

Parameters

*GraphicsScene** scene where the event took place

QKeyEvent * key that is pressed

Returns

void

6.28.2.26 void Tinkercell::GraphicsScene::keyPressEvent (QKeyEvent * *keyEvent*)
[protected, virtual]

when key is pressed

when key is pressed Precondition: None Postcondition: None

Parameters

QKeyEvent * key event

Returns

void

Parameters

key event

Returns

void

6.28.2.27 void Tinkercell::GraphicsScene::keyReleased (GraphicsScene * *scene*, QKeyEvent *)
[signal]

signals whenever a key is released

Parameters

*GraphicsScene** scene where the event took place

QKeyEvent * key that is released

Returns

void

6.28.2.28 void Tinkercell::GraphicsScene::keyReleaseEvent (QKeyEvent * *keyEvent*) [protected, virtual]

when key is released

when key is released Precondition: None Postcondition: None

Parameters

QKeyEvent * key event

Returns

void

Parameters

key event

Returns

void

6.28.2.29 QPointF & Tinkercell::GraphicsScene::lastPoint () [virtual]

Returns the point where mouse was clicked last on the scene coordinates.

Returns the point where mouse was clicked last Precondition: None Postcondition: None.

Parameters

void

Returns

QPointF& ref to last clicked point on the scene

Parameters

void

Returns

ref to last clicked point

6.28.2.30 QPoint & Tinkercell::GraphicsScene::lastScreenPoint () [virtual]

Returns the point where mouse was clicked last on the screen coordinates.

Returns the point where mouse was clicked last Precondition: None Postcondition: None.

Parameters

void

Returns

QPointF& ref to last clicked point on the screen

Parameters*void***Returns**

ref to last clicked point

6.28.2.31 `QRect TinkerCell::GraphicsScene::mapToWidget (QRectF rect = QRectF (0, 0, 0, 0)) const [virtual, slot]`

map a rect from the scene coordinates to the view coordinates

Parameters*QRectF* if left blank, the visible rect will be used**Returns***void*

6.28.2.32 `void TinkerCell::GraphicsScene::mouseDoubleClicked (GraphicsScene * scene, QPointF point, QGraphicsItem *, Qt::MouseButton, Qt::KeyboardModifiers modifiers) [signal]`

emits event when mouse is double clicked

Parameters*GraphicsScene** scene where the event took place*point* where mouse is clicked*modifier* keys being used when mouse clicked**Returns***void*

6.28.2.33 `void TinkerCell::GraphicsScene::mouseDoubleClickEvent (QGraphicsSceneMouseEvent * mouseEvent) [protected, virtual]`

when mouse is double clicked, the item at the position is added to selected list and moving list

emits signal when mouse is double clicked Precondition: None Postcondition: None

Parameters*QGraphicsSceneMouseEvent* * mouse event**Returns***void***Parameters***mouse* event

Returns

void

6.28.2.34 void Tinkercell::GraphicsScene::mouseDragged (GraphicsScene * *scene*, QPointF *from*, QPointF *to*, Qt::MouseButton, Qt::KeyboardModifiers *modifiers*) [signal]

signals whenever mouse is dragged from one point to another

Parameters

*GraphicsScene** scene where the event took place

QPointF point where mouse is clicked first

QPointF point where mouse is released

Qt::MouseButton button being pressed

Qt::KeyboardModifiers modifier keys being used when mouse clicked

Returns

void

6.28.2.35 void Tinkercell::GraphicsScene::mouseMoved (GraphicsScene * *scene*, QGraphicsItem * *item*, QPointF *point*, Qt::MouseButton, Qt::KeyboardModifiers *modifiers*, QList< QGraphicsItem * > &) [signal]

signals whenever mouse moves, and indicates whether it is on top of an item

Parameters

*GraphicsScene** scene where the event took place

*QGraphicsItem** pointer to item that mouse is on top of

QPointF point where mouse is clicked

Qt::MouseButton button being pressed

Qt::KeyboardModifiers modifier keys being used when mouse clicked

QList<QGraphicsItem>&* list of items that are being moved with the mouse

Returns

void

6.28.2.36 void Tinkercell::GraphicsScene::mouseMoveEvent (QGraphicsSceneMouseEvent * *mouseEvent*) [protected, virtual]

when mouse is moving, all items in moving list are moved

when mouse is moving, all items in moving list are moved Precondition: None Postcondition: None

Parameters

QGraphicsSceneMouseEvent * mouse event

Returns

void

Parameters

mouse event

Returns

void

6.28.2.37 void TinkerCell::GraphicsScene::mouseOnTopOf (GraphicsScene * *scene*, QGraphicsItem * *item*, QPointF *point*, Qt::KeyboardModifiers *modifiers*, QList< QGraphicsItem * > &) [signal]

signals whenever mouse is on top of an item

Parameters

*GraphicsScene** scene where the event took place

*QGraphicsItem** pointer to item that mouse is on top of

QPointF point where mouse is clicked

Qt::KeyboardModifiers modifier keys being used when mouse clicked

QList<QGraphicsItem>&* list of items that are being moved with the mouse

Returns

void

6.28.2.38 void TinkerCell::GraphicsScene::mousePressed (GraphicsScene * *scene*, QPointF *point*, Qt::MouseButton, Qt::KeyboardModifiers *modifiers*) [signal]

signals whenever an empty node of the screen is clicked

Parameters

*GraphicsScene** scene where the event took place

QPointF point where mouse is clicked

Qt::MouseButton which button was pressed

Qt::KeyboardModifiers modifier keys being used when mouse clicked

Returns

void

6.28.2.39 void Tinkercell::GraphicsScene::mousePressEvent (QGraphicsSceneMouseEvent * *mouseEvent*) [protected, virtual]

when mouse is pressed, the item at the position is added to selected list and moving list

when mouse is pressed, the item at the position is added to selected list and moving list Precondition: None
Postcondition: None

Parameters

QGraphicsSceneMouseEvent * mouse event

Returns

void

Parameters

mouse event

Returns

void

6.28.2.40 void Tinkercell::GraphicsScene::mouseReleased (GraphicsScene * *scene*, QPointF *point*, Qt::MouseButton, Qt::KeyboardModifiers *modifiers*) [signal]

signals whenever an empty node of the screen is clicked

Parameters

*GraphicsScene** *scene* where the event took place

QPointF *point* where mouse is clicked

Qt::MouseButton which button was pressed

Qt::KeyboardModifiers modifier keys being used when mouse clicked

Returns

void

6.28.2.41 void Tinkercell::GraphicsScene::mouseReleaseEvent (QGraphicsSceneMouseEvent * *mouseEvent*) [protected, virtual]

when mouse is released, moving list is cleared

when mouse is released, moving list is cleared Precondition: None Postcondition: None

Parameters

QGraphicsSceneMouseEvent * mouse event

Returns

void

Parameters

mouse event

Returns

void

6.28.2.42 void Tinkercell::GraphicsScene::move (QGraphicsItem * *item*, const QPointF & *distance*) [virtual, slot]

a simple move operation that also adds undo command to history window and emits associated signal(s)

a simple move operation with undo

Parameters

QGraphicsItem * *item* to move

QPointF *distance* to move the item

Returns

void

6.28.2.43 void Tinkercell::GraphicsScene::move (const QList< QGraphicsItem * > & *items*, const QPointF & *distance*) [virtual, slot]

a simple move operation that also adds undo command to history window and emits associated signal(s)

a simple move operation with undo

Parameters

QList<QGraphicsItem>&* *items* to move

QPointF *distance* to move the items (same for all items)

Returns

void

6.28.2.44 void Tinkercell::GraphicsScene::move (const QList< QGraphicsItem * > & *items*, const QList< QPointF > & *distance*) [virtual, slot]

a simple move operation that also adds undo command to history window and emits associated signal(s)

a simple move operation with undo

Parameters

QList<QGraphicsItem>&* *items* to move

QList<QPointF>& *distance* to move the items specified for each item

Returns

void

6.28.2.45 `QList< QGraphicsItem * > & Tinkercell::GraphicsScene::moving () [virtual]`

Returns the list of pointers to items that are currently being moved.

Returns the list of pointers to items that are currently being moved Precondition: None Postcondition: None.

Parameters

void

Returns

`QList<QGraphicsItem*>&` list of pointers to moving items

Parameters

void

Returns

list of pointers to moving items

6.28.2.46 `void Tinkercell::GraphicsScene::parentItemChanged (QGraphicsScene * scene, const QList< QGraphicsItem * > & items, const QList< QGraphicsItem * > & parents) [signal]`

signals whenever item parents are changed

Parameters

GraphicsScene * scene where the event took place

QList<QGraphicsItem>&* items

QList<QGraphicsItem>&* new parents

Returns

void

6.28.2.47 `void Tinkercell::GraphicsScene::popIn () [virtual, slot]`

calls main window's popIn

Returns

void

6.28.2.48 `void Tinkercell::GraphicsScene::popOut () [virtual, slot]`

calls main window's popOut

Returns

void

6.28.2.49 void Tinkercell::GraphicsScene::populateContextMenu () [protected, virtual]

populate the context menu using selected items' tools actions

Returns

void

6.28.2.50 void Tinkercell::GraphicsScene::print (QPaintDevice * *printer*, const QRectF & *rect* = QRectF()) [virtual]

send everything on the screen to a printer

prints the current scene

Parameters

QPaintDevice * *printer*

QRectF region to print

Returns

void

6.28.2.51 void Tinkercell::GraphicsScene::remove (const QString & *name*, QGraphicsItem * *item*) [virtual, slot]

this command performs an removal and also adds undo command to history window and emits associated signal(s)

this command performs an removal and allows redo/undo of that removal

6.28.2.52 void Tinkercell::GraphicsScene::remove (const QString & *name*, const QList< QGraphicsItem * > & *items*) [virtual, slot]

this command performs an removal and also adds undo command to history window and emits associated signal(s)

this command performs an removal and allows redo/undo of that removal

6.28.2.53 void Tinkercell::GraphicsScene::sceneRightClick (GraphicsScene * *scene*, QGraphicsItem * *item*, QPointF *point*, Qt::KeyboardModifiers *modifiers*) [signal]

signals whenever right click is made on an item or scene

Parameters

*GraphicsScene** *scene* where the event took place

*QGraphicsItem** pointer to item that mouse is clicked on

QPointF point where mouse is clicked

Qt::KeyboardModifiers modifier keys being used when mouse clicked

Returns

void

6.28.2.54 void Tinkercell::GraphicsScene::select (const QList< QGraphicsItem * > & *item*)
[virtual]

select items (does not deselect previously selected items)

select items

Parameters

QList<QGraphicsItem>&* items to select

Returns

void

6.28.2.55 void Tinkercell::GraphicsScene::select (QGraphicsItem * *item*) [virtual]

select one item (does not deselect other items)

select items

Parameters

*QGraphicsItem** item to select

Returns

void

6.28.2.56 QList< QGraphicsItem * > & Tinkercell::GraphicsScene::selected () [virtual]

Returns the list of pointers to items that are currently selected.

Returns the list of pointers to items that are currently selected Precondition: None Postcondition: None.

Parameters

void

Returns

QList<QGraphicsItem*>& list of pointers to selected items

Parameters

void

Returns

list of pointers to selected items

6.28.2.57 QRectF TinkerCell::GraphicsScene::selectedRect () [virtual]

Returns a rectangle that includes all the selected items.

Returns a rectangle that includes all the selected items Precondition: None Postcondition: None.

Parameters

void

Returns

QRectF bounding rect for selected items

Parameters

void

Returns

bounding rect for selected items

6.28.2.58 void TinkerCell::GraphicsScene::setBrush (const QString & name, const QList< QGraphicsItem * > & items, const QList< QBrush > & to) [virtual, slot]

this command changes the brush of an item and also adds undo command to history window and emits associated signal(s)

this command changes the brush of an item

6.28.2.59 void TinkerCell::GraphicsScene::setBrushAndPen (const QString & name, QGraphicsItem * item, const QBrush & brush, const QPen & pen) [virtual, slot]

this command changes the pen and/or brush of an item and also adds undo command to history window and emits associated signal(s)

this command changes the pen of an item

6.28.2.60 void TinkerCell::GraphicsScene::setBrushAndPen (const QString & name, const QList< QGraphicsItem * > & items, const QList< QBrush > & brushes, const QList< QPen > & pens) [virtual, slot]

this command changes the pen and/or brush of an item and also adds undo command to history window and emits associated signal(s)

this command changes the pen of an item

6.28.2.61 void TinkerCell::GraphicsScene::setGridSize (int sz = 100) [virtual]

set the grid size. If > 0, grid will be enabled. If 0, grid will be disabled

Parameters

double grid size (0 will disable grid)

Returns

void

6.28.2.62 void Tinkercell::GraphicsScene::setParentItem (const QString & *name*, const QList< QGraphicsItem * > & *items*, const QList< QGraphicsItem * > & *newParents*)
[virtual, slot]

this command changes the parent of an item and also adds undo command to history window and emits associated signal(s)

this command changes the parent of an item

6.28.2.63 void Tinkercell::GraphicsScene::setParentItem (const QString & *name*, const QList< QGraphicsItem * > & *items*, QGraphicsItem * *newParent*) [virtual, slot]

this command changes the parent of an item and also adds undo command to history window and emits associated signal(s)

this command changes the parent of an item

6.28.2.64 void Tinkercell::GraphicsScene::setParentItem (const QString & *name*, QGraphicsItem * *item*, QGraphicsItem * *newParent*) [virtual, slot]

this command changes the parent of an item and also adds undo command to history window and emits associated signal(s)

this command changes the parent of an item

6.28.2.65 void Tinkercell::GraphicsScene::setPen (const QString & *name*, const QList< QGraphicsItem * > & *items*, const QList< QPen > & *to*) [virtual, slot]

this command changes the pen of an item and also adds undo command to history window and emits associated signal(s)

this command changes the pen of an item

6.28.2.66 void Tinkercell::GraphicsScene::setPen (const QString & *name*, QGraphicsItem * *item*, const QPen & *to*) [virtual, slot]

this command changes the pen of an item and also adds undo command to history window and emits associated signal(s)

this command changes the pen of an item

6.28.2.67 void Tinkercell::GraphicsScene::snapToGrid (QGraphicsItem * *item*) [virtual]

snap the node item to the grid

Parameters

*NodeGraphicsItem**

Returns

void

6.28.2.68 void TinkerCell::GraphicsScene::transform (const QString & *name*, QGraphicsItem * *item*, const QPointF & *sizechange*, qreal *anglechange* = 0.0, bool *VFlip* = *false*, bool *HFlip* = *false*) [virtual, slot]

this command changes the size, angle, and orientation of an item and also adds undo command to history window and emits associated signal(s)

this command changes the size, angle, and orientation of an item

6.28.2.69 void TinkerCell::GraphicsScene::transform (const QString & *name*, const QList< QGraphicsItem * > & *items*, const QList< QPointF > & *sizechange*, const QList< qreal > & *anglechange* = QList< qreal > (), const QList< bool > & *verticalFlip* = QList< bool > (), const QList< bool > & *horizontalFlip* = QList< bool > ()) [virtual, slot]

this command changes the size, angle, and orientation of an item and also adds undo command to history window and emits associated signal(s)

this command changes the size, angle, and orientation of an item

6.28.2.70 QRectF TinkerCell::GraphicsScene::visibleRegion () const [virtual]

Returns the currently visible window from the current graphics view.

Returns the currently visible window.

Parameters

void

Returns

QRectF rectangle

Parameters

void

Returns

rectangle

6.28.2.71 void TinkerCell::GraphicsScene::zoom (qreal *scaleFactor*) [virtual, slot]

zoom in or out

zoom

Parameters

scale factor (< 1 means zoom out)

Returns

void

Parameters

scale factor

Returns

void

6.28.2.72 void Tinkercell::GraphicsScene::zoomIn () [virtual, slot]

zoom in (zoom with 1.5)

zoom in

Returns

void

Parameters

scale factor

Returns

void

6.28.2.73 void Tinkercell::GraphicsScene::zoomOut () [virtual, slot]

zoom out (zoom with 0.75)

zoom out

Parameters

scale factor

Returns

void

6.28.2.74 qreal Tinkercell::GraphicsScene::ZValue () [virtual]

top Z value

top Z value Precondition: None Postcondition: None

Returns

double

The documentation for this class was generated from the following files:

- GraphicsScene.h
- GraphicsScene.cpp

6.29 TinkerCell::GraphicsView Class Reference

[GraphicsView](#) class that is used to view the contents of a [GraphicsScene](#). The class inherits from [QGraphicsView](#).

```
#include <GraphicsView.h>
```

Signals

- void [itemsDropped](#) ([GraphicsScene](#) *, const QString &, const QPointF &)
signal is emitted when some object OTHER than files are dropped on the canvas

Protected Member Functions

- virtual void [drawBackground](#) (QPainter *painter, const QRectF &rect)
draw background
- virtual void [drawForeground](#) (QPainter *painter, const QRectF &rect)
draw foreground
- virtual void [dropEvent](#) (QDropEvent *)
drag and drop
- virtual void [dragEnterEvent](#) (QDragEnterEvent *event)
drag and drop
- virtual void [dragMoveEvent](#) (QDragMoveEvent *event)
drag and drop
- virtual void [wheelEvent](#) (QWheelEvent *event)
mouse wheel event
- virtual void [scrollContentsBy](#) (int dx, int dy)
scroll event
- virtual void [mousePressEvent](#) (QMouseEvent *event)
mouse event. sets the currentGraphicsView for [NetworkWindow](#)
- virtual void [keyPressEvent](#) (QKeyEvent *event)
mouse event. sets the currentGraphicsView for [NetworkWindow](#)
- virtual void [mouseMoveEvent](#) (QMouseEvent *event)
when moved using right button or ctrl, mode switches to drag

Friends

- class **GraphicsScene**
- class **NetworkWindow**
- class **NetworkHandle**
- class **MainWindow**

6.29.1 Detailed Description

GraphicsView class that is used to view the contents of a **GraphicsScene**. The class inherits from **QGraphicsView**.

The documentation for this class was generated from the following files:

- GraphicsView.h
- GraphicsView.cpp

6.30 TinkerCell::HistoryWindow Class Reference

This is a small class extending **QUndoView** that manages a stack of undo commands.

```
#include <HistoryWindow.h>
```

Public Slots

- void **undo** ()
- void **redo** ()
- void **push** (**QUndoCommand** *command)

6.30.1 Detailed Description

This is a small class extending **QUndoView** that manages a stack of undo commands.

The documentation for this class was generated from the following files:

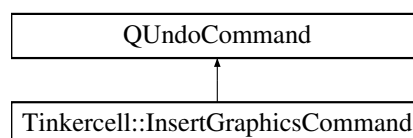
- HistoryWindow.h
- HistoryWindow.cpp

6.31 TinkerCell::InsertGraphicsCommand Class Reference

this command performs an insert and allows redo/undo of that insert

```
#include <UndoCommands.h>
```

Inheritance diagram for **TinkerCell::InsertGraphicsCommand**:



Public Member Functions

- [InsertGraphicsCommand](#) (const QString &name, [GraphicsScene](#) *scene, QGraphicsItem *item, bool checkNames=true)
constructor
- [InsertGraphicsCommand](#) (const QString &name, [GraphicsScene](#) *scene, const QList< QGraphicsItem * > &items, bool checkNames=true)
constructor
- void [redo](#) ()
redo the change
- void [undo](#) ()
undo the change
- virtual [~InsertGraphicsCommand](#) ()
destructor

6.31.1 Detailed Description

this command performs an insert and allows redo/undo of that insert

6.31.2 Constructor & Destructor Documentation

6.31.2.1 TinkerCell::InsertGraphicsCommand::InsertGraphicsCommand (const QString & name, GraphicsScene * scene, QGraphicsItem * item, bool checkNames = true)

constructor

Parameters

- QString* name of command
*GraphicsScene** where change happened
*QGraphicsItem** item that is inserted
bool check for uniqueness of names before inserting (default = true)

6.31.2.2 TinkerCell::InsertGraphicsCommand::InsertGraphicsCommand (const QString & name, GraphicsScene * scene, const QList< QGraphicsItem * > & items, bool checkNames = true)

constructor

Parameters

- QString* name of command
*GraphicsScene** where change happened
QList<QGraphicsItem>&* items that are inserted

bool check for uniqueness of names before inserting (default = true)

The documentation for this class was generated from the following files:

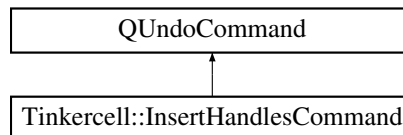
- UndoCommands.h
- UndoCommands.cpp

6.32 Tinkercell::InsertHandlesCommand Class Reference

this command inserts new handles to a [NetworkHandle](#)

```
#include <UndoCommands.h>
```

Inheritance diagram for Tinkercell::InsertHandlesCommand:



Public Member Functions

- [InsertHandlesCommand](#) (TextEditor *, const QList< [ItemHandle](#) * > &, bool checkNames=true)
constructor
- [InsertHandlesCommand](#) (TextEditor *, [ItemHandle](#) *, bool checkNames=true)
constructor
- [~InsertHandlesCommand](#) ()
destructor: deletes all text items and their handles (if not containing any graphics items)
- void [redo](#) ()
redo the change
- void [undo](#) ()
undo the change

6.32.1 Detailed Description

this command inserts new handles to a [NetworkHandle](#)

6.32.2 Constructor & Destructor Documentation

6.32.2.1 Tinkercell::InsertHandlesCommand::InsertHandlesCommand (TextEditor * *textEditor*, const QList< ItemHandle * > & *list*, bool *checkNames* = *true*)

constructor

Parameters

*NetworkHandle** window where items are inserted

QList<ItemHandle>* new items

bool check for uniqueness of names before inserting

6.32.2.2 Tinkercell::InsertHandlesCommand::InsertHandlesCommand (*TextEditor * textEditor*, *ItemHandle * h*, *bool checkNames = true*)

constructor

Parameters

*NetworkHandle** window where items are inserted

*ItemHandle** new item

bool check for uniqueness of names before inserting

The documentation for this class was generated from the following files:

- UndoCommands.h
- UndoCommands.cpp

6.33 Tinkercell::ItemData Class Reference

This class is used to store information about nodes or connections. It contains a hashtable of data tables, which is used by different tools to store specific data. The versions queue can be used to keep previous versions of the data.

```
#include <ItemHandle.h>
```

Friends

- class **ItemHandle**

6.33.1 Detailed Description

This class is used to store information about nodes or connections. It contains a hashtable of data tables, which is used by different tools to store specific data. The versions queue can be used to keep previous versions of the data.

The documentation for this class was generated from the following files:

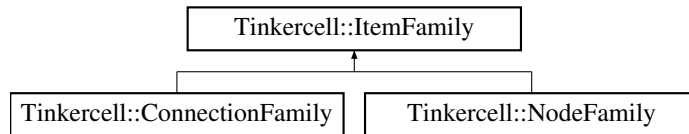
- ItemHandle.h
- ItemHandle.cpp

6.34 Tinkercell::ItemFamily Class Reference

This class defines the family of a node or connection. The class contains the icon for the family, family name, and minimal data that defines the family. Each family has a name, which is internally converted to an integer (ID) The ID is used to perform isA checks, thus avoiding repeated string matches.

```
#include <ItemFamily.h>
```

Inheritance diagram for Tinkercell::ItemFamily:



Public Member Functions

- virtual QString **name** () const
name of this family
- virtual void **setName** (const QString &)
set name of this family
- virtual bool **isA** (const QString &) const
indicates whether or not the given string is the name of this family or any of its parent families
- virtual bool **isA** (const ItemFamily *) const
indicates whether or not the given family is the name of this family or any of its parent families
- virtual bool **isParentOf** (const QString &) const
indicates whether or not the given string is the name of this family or any of its child families
- virtual bool **isParentOf** (const ItemFamily *) const
indicates whether or not the given family is the name of this family or any of its child families
- virtual ItemFamily * **root** () const
get the top-most family
- virtual bool **isRelatedTo** (const ItemFamily *) const
checks if the given family shares its root family with this family
- virtual ItemFamily * **parent** () const
get the parent for this family. If there are more than one parents, returns the first
- virtual QList< ItemFamily * > **parents** () const
get all the parents for this family.
- virtual QList< ItemFamily * > **children** () const
get all the families that inherit directly from this family

- virtual QList< [ItemFamily](#) * > [allChildren](#) () const
get all the families that inherit from this family. the list will be ordered in a breadth-first ordering
- [ItemFamily](#) (const QString &name=QString())
constructor.
- virtual ~[ItemFamily](#) ()
destructor.

Public Attributes

- QString [description](#)
description of this family
- QStringList [restrictions](#)
restrictions that apply to this family
- QList< [Unit](#) > [measurementUnitOptions](#)
the possible options for measurement name and unit for items in this family
- [Unit](#) [measurementUnit](#)
the measurement name and unit for items in this family
- QHash< QString, qreal > [numericalAttributes](#)
the list of numerical attributes that are common to all members of this family
- QHash< QString, QString > [textAttributes](#)
the list of string attributes that are common to all members of this family
- QList< QGraphicsItem * > [graphicsItems](#)
the default set of graphics items used to represent items of this family
- QPixmap [pixmap](#)
the icon representing this family

Protected Member Functions

- virtual bool [isA](#) (int [ID](#)) const
indicates whether or not the given family ID is the name of this family or any of its parent families

Protected Attributes

- int [type](#)
used for casting between different sub-classes

- QString [_name](#)
name of this family
- int [ID](#)
the ID for this family. It is used for quick equality checks (instead of using strings)

Static Protected Attributes

- static QStringList [ALLNAMES](#)
all family names. This list's length is used to assign the next ID
- static QList< const [ItemFamily](#) * > [ALLFAMILIES](#)
all families by index
- static QHash< QString, int > [NAMETOID](#)
the hash stores names for each ID

Friends

- class [NodeFamily](#)
- class [ConnectionFamily](#)

6.34.1 Detailed Description

This class defines the family of a node or connection. The class contains the icon for the family, family name, and minimal data that defines the family. Each family has a name, which is internally converted to an integer (ID) The ID is used to perform isA checks, thus avoiding repeated string matches.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 TinkerCell::ItemFamily::ItemFamily (const QString & name = QString())

constructor.

Parameters

QString name

6.34.3 Member Function Documentation

6.34.3.1 QList< ItemFamily * > TinkerCell::ItemFamily::allChildren () const [virtual]

get all the families that inherit from this family. the list will be ordered in a breadth-first ordering

Returns

QList<ItemFamily*>

The documentation for this class was generated from the following files:

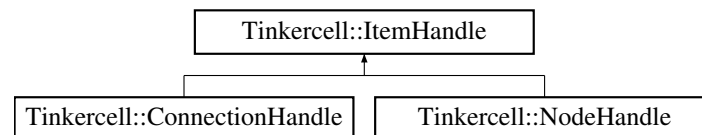
- ItemFamily.h
- ItemFamily.cpp

6.35 Tinkercell::ItemHandle Class Reference

The [ItemHandle](#) represents a complete object in the network, whether it is a node or a connection. The [ItemHandle](#) contains the name of the object and pointers to all the QGraphicsItems that are used to represent the object. Tools associated with the object can be stored within the [ItemHandle](#) as well. The [ItemHandle](#) can also optionally contain an [ItemFamily](#), which can be used to distinguish different types of nodes or connections, if needed. Each [ItemHandle](#) can contain one parent. Several functions are available for conveniently getting the parents and children of an [ItemHandle](#).

```
#include <ItemHandle.h>
```

Inheritance diagram for Tinkercell::ItemHandle:



Public Member Functions

- [ItemHandle](#) (const QString &name=QString())
default constructor
- [ItemHandle](#) (const [ItemHandle](#) &)
copy constructor
- virtual [ItemHandle](#) & operator= (const [ItemHandle](#) &)
operator =
- virtual ~[ItemHandle](#) ()
destructor -- does nothing
- virtual [ItemHandle](#) * clone () const
clone the data and lists
- virtual [ItemFamily](#) * family () const
family that this items belongs in. Used for characterizing the nodes and connections.
- virtual void setFamily ([ItemFamily](#) *, bool useCommand=true)
set the family that this items belongs in.
- virtual bool isA (const [ItemFamily](#) *family) const
determines whether this handle belongs to the speicific family.

- virtual bool **isA** (const QString &family) const
determines whether this handle belongs to the speicific family.
- virtual QString **fullName** (const QString &sep=QString(".")) const
The full name includes all the parent names appended using a dot.
- virtual void **setParent** (ItemHandle *parent, bool useCommand=true)
Set the parent for this handle.
- virtual void **rename** (const QString &)
set name of this handle and also adds undo command to history window and emits associated signal(s)
- virtual void **changeData** (const QString &hashstring, const NumericalDataTable *newdata)
change numerical data table and also adds undo command to history window and emits associated signal(s)
- virtual void **changeData** (const QString &hashstring, const TextDataTable *newdata)
change text data table and also adds undo command to history window and emits associated signal(s)
- virtual ItemHandle * **root** (const QString &family=QString("")) const
get the top-level handle such that it is of the specified family. If no family is specified, then gets the top-level handle
- virtual ItemHandle * **parentOfFamily** (const QString &family) const
get the bottom-most parent handle such that it is of the specified family. If no family is specified, then gets the top-level handle
- virtual bool **isChildOf** (ItemHandle *handle) const
checks if an item is the parent or parent's parent, or parent's parent's parent, etc. Note: self->isChildOf(self) is false
- virtual int **depth** () const
counts the number of parents that have to be traversed in order to reach the root handle. If this handle has no parents, the values returned is 0. If its parent has no parent, then the value is 1, and so on.
- virtual QList< QGraphicsItem * > **allGraphicsItems** () const
gets the graphics items belonging to this handle and all child handles
- virtual QList< ItemHandle * > **allChildren** () const
gets the all child handles and their child handles
- QStringList **numericalDataNames** () const
all the numerical data table names
- QStringList **textDataNames** () const
all the numerical text table names
- bool **hasNumericalData** (const QString &name) const
does this handle have a numerical data table with this name?
- bool **hasTextData** (const QString &name) const

does this handle have a text data table with this name?

- `qreal numericalData (const QString &name, int row=0, int column=0) const`
gets a numerical attribute with the given name, row, column
- `qreal numericalData (const QString &name, const QString &row, const QString &column=QString()) const`
gets a numerical attribute with the given name, row, column
- `QString textData (const QString &name, int row=0, int column=0) const`
gets a text attribute with the given name, row, column
- `QString textData (const QString &name, const QString &row, const QString &column=QString()) const`
gets a text attribute with the given name, row, column
- `qreal & numericalData (const QString &name, int row=0, int column=0)`
gets a reference to the numerical attribute with the given name, row, column
- `qreal & numericalData (const QString &name, const QString &row, const QString &column=QString())`
gets a reference to the numerical attribute with the given name, row, column
- `QString & textData (const QString &name, int row=0, int column=0)`
gets a reference to the text attribute with the given name, row, column
- `QString & textData (const QString &name, const QString &row, const QString &column=QString())`
gets a reference to the text attribute with the given name, row, column
- `NumericalDataTable & numericalDataTable (const QString &name)`
gets reference to a numerical table with the given name. Makes the table if needed
- `TextDataTable & textDataTable (const QString &name)`
gets reference to a text table with the given name. Makes the table if needed

Public Attributes

- `QString name`
name of this item
- `QList< QGraphicsItem * > graphicsItems`
list of graphical items used to draw this handle
- `QList< Tool * > tools`
list of tools associated with this handle
- `NetworkHandle * network`
the network that this item belongs in

- `ItemHandle * parent`
this handles immediate parent (main parent if there are more than one)
- `QList< ItemHandle * > children`
child handles that have this handle as a parent
- `int type`
type of this handle (sub-classes can specify type)

6.35.1 Detailed Description

The `ItemHandle` represents a complete object in the network, whether it is a node or a connection. The `ItemHandle` contains the name of the object and pointers to all the `QGraphicsItems` that are used to represent the object. Tools associated with the object can be stored within the `ItemHandle` as well. The `ItemHandle` can also optionally contain an `ItemFamily`, which can be used to distinguish different types of nodes or connections, if needed. Each `ItemHandle` can contain one parent. Several functions are available for conveniently getting the parents and children of an `ItemHandle`. Use `setHandle` and `getHandle` functions to get and set the handles for `QGraphicsItems`. Use `h->data->numericalData[string]` or `h->data->textData[string]` to get the `DataTable` with the particular name. Alternatively, `h->numericalData(string)` or `h->textData(string)` can be used to access the data conveniently.

The `SymbolsTable` is used to store all the handles in a network.

6.35.2 Constructor & Destructor Documentation

6.35.2.1 `Tinkercell::ItemHandle::ItemHandle (const QString & name = QString())`

default constructor

Parameters

`QString` name

6.35.3 Member Function Documentation

6.35.3.1 `QList< ItemHandle * > Tinkercell::ItemHandle::allChildren () const [virtual]`

gets the all child handles and their child handles

Returns

`QList<ItemHandle*>` list of handles

6.35.3.2 `QList< QGraphicsItem * > Tinkercell::ItemHandle::allGraphicsItems () const [virtual]`

gets the graphics items belonging to this handle and all child handles

Returns

QList<QGraphicsItem*> list of graphics items

6.35.3.3 int Tinkercell::ItemHandle::depth () const [virtual]

counts the number of parents that have to be traversed in order to reach the root handle. If this handle has no parents, the values returned is 0. If its parent has no parent, then the value is 1, and so on.

Returns

int

6.35.3.4 QString Tinkercell::ItemHandle::fullName (const QString & sep = QString(".") const [virtual]

The full name includes all the parent names appended using a dot.

Parameters

QString replace the dot with some other separator

6.35.3.5 bool Tinkercell::ItemHandle::hasNumericalData (const QString & name) const

does this handle have a numerical data table with this name?

Parameters

QString name of tool, e.g. "Numerical Attributes"

Returns

bool true = has a numerical table by this name. false = does not have a numerical table by this name

6.35.3.6 bool Tinkercell::ItemHandle::hasTextData (const QString & name) const

does this handle have a text data table with this name?

Parameters

QString name of tool, e.g. "Text Attributes"

Returns

bool true = has a text table by this name. false = does not have a text table by this name

6.35.3.7 bool Tinkercell::ItemHandle::isA (const ItemFamily * family) const [virtual]

determines whether this handle belongs to the speicific family.

Parameters

QString the family

6.35.3.8 bool Tinkercell::ItemHandle::isA (const QString & *family*) const [virtual]

determines whether this handle belongs to the speicific family.

Parameters

QString the family name

6.35.3.9 bool Tinkercell::ItemHandle::isChildOf (ItemHandle * *handle*) const [virtual]

checks if an item is the parent or parent's parent, or parent's parent's parent, etc. Note: self->isChildOf(self) is false

Parameters

*ItemHandle** parent handle

Returns

Boolean is child

6.35.3.10 qreal Tinkercell::ItemHandle::numericalData (const QString & *name*, const QString & *row*, const QString & *column* = *QString()*) const

gets a numerical attribute with the given name, row, column

Parameters

QString name of tool, e.g. "Numerical Attributes"

QString row name in data table

QString column name data table

Returns

double value

6.35.3.11 qreal Tinkercell::ItemHandle::numericalData (const QString & *name*, int *row* = 0, int *column* = 0) const

gets a numerical attribute with the given name, row, column

Parameters

QString name of tool, e.g. "Numerical Attributes"

int row in data table

int column in data table

Returns

double value

6.35.3.12 `qreal & TinkerCell::ItemHandle::numericalData (const QString & name, int row = 0, int column = 0)`

gets a reference to the numerical attribute with the given name, row, column

Parameters

QString name of tool, e.g. "Numerical Attributes"

int row in data table

int column in data table

Returns

double reference value

6.35.3.13 `qreal & TinkerCell::ItemHandle::numericalData (const QString & name, const QString & row, const QString & column = QString())`

gets a reference to the numerical attribute with the given name, row, column

Parameters

QString name of tool, e.g. "Numerical Attributes"

QString row name in data table

QString column name data table

Returns

double reference value

6.35.3.14 `QStringList TinkerCell::ItemHandle::numericalDataNames () const`

all the numerical data table names

Returns

QStringList

6.35.3.15 `DataTable< qreal > & TinkerCell::ItemHandle::numericalDataTable (const QString & name)`

gets reference to a numerical table with the given name. Makes the table if needed

Parameters

QString name of tool, e.g. "Numerical Attributes"

Returns

DataTable<double>& reference of table

6.35.3.16 `ItemHandle * Tinkercell::ItemHandle::parentOfFamily (const QString & family)
const [virtual]`

get the bottom-most parent handle such that it is of the specified family. If no family is specified, then gets the top-level handle

Parameters

*ItemHandle** the family name

6.35.3.17 `ItemHandle * Tinkercell::ItemHandle::root (const QString & family = QString("")
) const [virtual]`

get the top-level handle such that it is of the specified family. If no family is specified, then gets the top-level handle

Parameters

*ItemHandle** the family name

6.35.3.18 `void Tinkercell::ItemHandle::setParent (ItemHandle * parent, bool useCommand =
true) [virtual]`

Set the parent for this handle.

Parameters

*ItemHandle** parent

bool (optional) whether to call network's set parent command, which will update the history stack

*ItemHandle** parent handle

6.35.3.19 `QString Tinkercell::ItemHandle::textData (const QString & name, const QString &
row, const QString & column = QString()) const`

gets a text attribute with the given name, row, column

Parameters

QString name of tool, e.g. "Text Attributes"

QString row name in data table

QString column name data table

Returns

QString value

6.35.3.20 QString & TinkerCell::ItemHandle::textData (const QString & *name*, const QString & *row*, const QString & *column* = *QString()*)

gets a reference to the text attribute with the given name, row, column

Parameters

QString name of tool, e.g. "Text Attributes"

QString row name in data table

QString column name data table

Returns

QString& reference value

6.35.3.21 QString TinkerCell::ItemHandle::textData (const QString & *name*, int *row* = 0, int *column* = 0) const

gets a text attribute with the given name, row, column

Parameters

QString name of tool, e.g. "Text Attributes"

int row in data table

int column in data table

Returns

QString value

6.35.3.22 QString & TinkerCell::ItemHandle::textData (const QString & *name*, int *row* = 0, int *column* = 0)

gets a reference to the text attribute with the given name, row, column

Parameters

QString name of tool, e.g. "Text Attributes"

int row in data table

int column in data table

Returns

QString reference value

6.35.3.23 QStringList TinkerCell::ItemHandle::textDataNames () const

all the numerical text table names

Returns

QStringList

6.35.3.24 DataTable< QString > & Tinkercell::ItemHandle::textDataTable (const QString & name)

gets reference to a text table with the given name. Makes the table if needed

Parameters

QString name of tool, e.g. "Numerical Attributes"

Returns

TextDataTable& reference of table

The documentation for this class was generated from the following files:

- ItemHandle.h
- ItemHandle.cpp

6.36 Tinkercell::LineNumberArea Class Reference

Public Member Functions

- **LineNumberArea** ([CodeEditor](#) *editor)
- QSize **sizeHint** () const

Protected Member Functions

- void **paintEvent** (QPaintEvent *event)

The documentation for this class was generated from the following file:

- CodeEditor.h

6.37 Tinkercell::MainWindow Class Reference

[MainWindow](#) is the parent container for all the other widgets in TinkerCell The central widget in [Main-Window](#) is a tab widget. Each tab widget can hold a [GraphicsView](#) or a [TextEditor](#). One of the main roles of [MainWindow](#) is to serve as a signal/slot hub for Tools.

```
#include <MainWindow.h>
```

Public Types

- enum [TOOL_WINDOW_OPTION](#) { **DockWidget**, **TabWidget** }
this enum is used to determine how to place a widget when used in addToolWindow. DockWidget = tool window is placed into a dockable widget TabWidget = tool window is placed in an existing tab widget, if one exists
- enum [VIEW_MODE](#) { **TabView**, **WindowView** }
the types of views for multiple documents TabView = tabbed documents WindowView = each documents in a separate subwindow

Public Member Functions

- [MainWindow](#) (bool enableScene=true, bool enableText=true, bool views=true)
5-arg (optional) constructor allows disabling of text/graphics modes
- virtual void [allowMultipleViewModes](#) (bool)
allow or disallow changing between different views
- virtual [~MainWindow](#) ()
Destructor: delete all the graphics scenes.
- QDockWidget * [addToolWindow](#) (QWidget *tool, [TOOL_WINDOW_OPTION](#) option=DockWidget, Qt::DockWidgetArea initArea=Qt::RightDockWidgetArea, Qt::DockWidgetAreas allowedAreas=Qt::AllDockWidgetAreas, bool inMenu=true)
Add a new docking window to the main window. The name and icon are obtained using the widget's windowTitle and windowIcon, so be sure to set those before calling this function.
- void [addToViewMenu](#) (QWidget *tool)
place a show/hide action in the view menu for the given widget
- void [setCursor](#) (QCursor cursor)
set the cursor for all windows
- void [addTool](#) ([Tool](#) *tool)
add a new tool to the list of tools stored in the main window
- void [initializeMenus](#) (bool enableScene=true, bool enableText=true)
Initialize the basic menu (save, open, close, exit, etc.).
- void [setupNewThread](#) (QSemaphore *, QLibrary *)
This function is usually called from a new thread. This function allows all the plugins to add their functionalities to the C function pointer of the new thread.
- void [loadDynamicLibrary](#) (const QString &)
Load a new plugin (dll).
- QPair< QList< [ItemHandle](#) * >, QList< QGraphicsItem * > > [getItemsFromFile](#) (const QString &filename, [ItemHandle](#) *root=0)
get the items inside a file. Some tool must implement this function and connect to the getItemsFromFile signal. The Core library does not implement a read file function.
- [GraphicsScene](#) * [currentScene](#) () const
gets the current scene that is active
- [TextEditor](#) * [currentTextEditor](#) () const
gets the text editor that is active
- [NetworkWindow](#) * [currentWindow](#) () const
gets the current window that is active (each window contains either a scene or editor)
- [NetworkHandle](#) * [currentNetwork](#) () const

gets the current window that is active

- `QList< NetworkHandle * > networks () const`
gets all the windows in the main window
- `QUndoStack * historyStack () const`
the history stack of the current network.
- `QUndoView * historyWidget ()`
the history stack widget of the current window.
- `virtual Tool * tool (const QString &) const`
get a tool
- `virtual QList< Tool * > tools (const QString &category=QString()) const`
get all tools

Static Public Member Functions

- `static void RegisterDataTypes ()`
register all the TinkerCell data structures with Qt
- `static QString homeDir ()`
The TinkerCell user directory, which is User's Documents Folder/TinkerCell by default, but users may change this setting.
- `static QString tempDir ()`
The TinkerCell user temporary directory, which is <SYSTEM temp="" folder>="">/TinkerCell.

Public Attributes

- `QList< QWidget * > toolWindows`
the set of all windows inseted in the main window using addToolWindow
- `QMenu contextItemsMenu`
the context menu that is shown during right-click event on selected graphical items. Plugins can add new actions to this menu.
- `QMenu contextScreenMenu`
the context menu that is shown during right-click event on the scene. Plugins can add new actions to this menu.
- `QMenu contextSelectionMenu`
the context menu that is shown during right-click event on a text editor with text selected. Plugins can add new actions to this menu.
- `QMenu contextEditorMenu`

the context menu that is shown during right-click event on a text editor with no text selected. Plugins can add new actions to this menu.

- QMenu * [fileMenu](#)

The file menu. Plugins can add new actions to this menu.

- QMenu * [editMenu](#)

The edit menu. Plugins can add new actions to this menu.

- QMenu * [viewMenu](#)

The view menu. New docking windows are automatically added here.

- QMenu * [helpMenu](#)

The help menu.

- QMenu * [settingsMenu](#)

the menu for settings such as default plugins, Tinkercell home directory, etc.

- QMenu * [parsersMenu](#)

the menu for choosing one of the available parsers (will be 0 if there are no parsers)

- QToolBar * [toolBarBasic](#)

The tool bar that contains new, open, close, etc. actions.

- QToolBar * [toolBarEdits](#)

The tool bar that contains copy, paste, undo, etc.

- QToolBar * [toolBarForTools](#)

One of the initial tool bars which designated for tools that do not want to create a new toolbar.

Static Public Attributes

- static [TOOL_WINDOW_OPTION defaultToolWindowOption](#) = MainWindow::TabWidget

the default option to use for tools (optional)

- static [TOOL_WINDOW_OPTION defaultHistoryWindowOption](#) = MainWindow::TabWidget

the default option to use for history window

- static [TOOL_WINDOW_OPTION defaultConsoleWindowOption](#) = MainWindow::DockWidget

the default option to use for console window

- static QString [PROJECTWEBSITE](#) = QObject::tr("www.tinkercell.com")

the project website

- static QString [ORGANIZATIONNAME](#) = QObject::tr("TinkerCell")

the project organization name

- static QString [PROJECTNAME](#) = QObject::tr("TinkerCell")

the project name

- static QString [CPP_ENTRY_FUNCTION](#) = QObject::tr("loadTCTool")
the default function that is loaded in C++ plugins
- static QString [C_ENTRY_FUNCTION](#) = QObject::tr("tc_main")
the default function that is loaded in C plugins
- static QString [PROJECT_VERSION](#) = QObject::tr("0.0.0")
the default project version
- static QString [PROGRAM_MODE](#)
an optional string that can be used to change the mode of the application. The meaning of this variable depends on the purpose of the application. Empty by default.
- static QStringList [OPEN_FILE_EXTENSIONS](#)
the default file extensions that can be opened
- static QStringList [SAVE_FILE_EXTENSIONS](#)
the default file extensions that can be saved

Friends

- class **NetworkWindow**
- class **NetworkHandle**
- class **GraphicsScene**
- class **TextEditor**
- class **GraphicsView**

signals

- static QString [previousFileName](#)
stores the last opened directory
- static QHash< void *, bool > [invalidPointers](#)
stores list of all pointers that have been deleted (to prevent double-deletions)
- bool [allowViewModeToChange](#)
allowed views
- QHash< QString, QLibrary * > [dynamicallyLoadedLibraries](#)
the loaded dynamic libraries indexed by file name
- [ConsoleWindow](#) * [consoleWindow](#)
the general window for command, errors, and messages
- QTabWidget * [tabWidget](#)
the central multi-document interface widget

- QList< [NetworkHandle](#) * > [allNetworks](#)
the list of all network windows
- QTabWidget * [toolsTabWidget](#)
the optional tool box that will only appear if one of the plug-ins uses the tab widget argument in the addToolWindow call
- [HistoryWindow](#) [historyWindow](#)
history view, not the stack itself. The stack is stored within each [NetworkHandle](#)
- [NetworkWindow](#) * [currentNetworkWindow](#)
keep pointer to last selected window. Used by windowChanged signal
- QHash< QString, [Tool](#) * > [toolsHash](#)
all the tools (plug-ins) are stored here, indexed by their names
- QHash< QString, [Tool](#) * > [toolsHashByCategory](#)
this is a multiple hash. All the tool are stored here indexed by their category names (if they have a category)
- bool [isValidHandlePointer](#) (void *p)
checks if the given address belongs to a handle
- void [toolAboutToBeLoaded](#) ([Tool](#) *tool, bool *shouldLoad)
a new tool is about to be added. This signal can be used to prevent the tool from being added
- void [historyChanged](#) (int i=0)
one of more changed have occurred in the history window of the current scene
- void [funtionPointersToMainThread](#) (QSemaphore *, QLibrary *)
used internally by [MainWindow](#) in order to move from a thread to the main thread
- void [toolLoaded](#) ([Tool](#) *tool)
signals when a new tool (plugin) is loaded
- void [setupFunctionPointers](#) (QLibrary *)
signals when a new FuntionToSignal is constructed
- void [networkClosing](#) ([NetworkHandle](#) *, bool *)
signals when a network is going to close
- void [networkClosed](#) ([NetworkHandle](#) *)
signals after a window is closed
- void [prepareNetworkForSaving](#) ([NetworkHandle](#) *, bool *)
signals when a tool is about to save a network
- void [networkSaved](#) ([NetworkHandle](#) *)
signals when a tool has saved the network in a file
- void [saveNetwork](#) (const QString &filename)

signals when user selects a file to save the current network to

- void [loadNetwork](#) (const QString &filename)
signals when user selects a file to open in the current network
- void [getItemsFromFile](#) (QList< [ItemHandle](#) * > &, QList< QGraphicsItem * > &, const QString &filename, [ItemHandle](#) *root)
signal sent to a tool so that the tool can get the items inside a file
- void [networkLoaded](#) ([NetworkHandle](#) *)
signals informs that the current network has just loaded a new Network
- void [networkOpened](#) ([NetworkHandle](#) *)
signals whenever the new network is opened
- void [windowChanged](#) ([NetworkWindow](#) *, [NetworkWindow](#) *)
signals whenever the current window changes
- void [itemsSelected](#) ([GraphicsScene](#) *scene, const QList< QGraphicsItem * > &items, QPointF point, Qt::KeyboardModifiers modifiers)
signals whenever a new item is selected (item can be sub-item, not top-level)
- void [mousePressed](#) ([GraphicsScene](#) *scene, QPointF point, Qt::MouseButton, Qt::KeyboardModifiers modifiers)
signals whenever an empty node of the screen is clicked
- void [mouseReleased](#) ([GraphicsScene](#) *scene, QPointF point, Qt::MouseButton, Qt::KeyboardModifiers modifiers)
signals whenever an empty node of the screen is clicked
- void [mouseDoubleClicked](#) ([GraphicsScene](#) *scene, QPointF point, QGraphicsItem *, Qt::MouseButton, Qt::KeyboardModifiers modifiers)
emits event when mouse is double clicked
- void [mouseDragged](#) ([GraphicsScene](#) *scene, QPointF from, QPointF to, Qt::MouseButton, Qt::KeyboardModifiers modifiers)
signals whenever mouse is dragged from one point to another
- void [itemsAboutToBeMoved](#) ([GraphicsScene](#) *scene, QList< QGraphicsItem * > &item, QList< QPointF > &distance, QList< [QUndoCommand](#) * > &)
signals whenever items are going to be moved (each item is the top-most item)
- void [itemsMoved](#) ([GraphicsScene](#) *scene, const QList< QGraphicsItem * > &item, const QList< QPointF > &distance)
signals whenever items are being moved (each item is the top-most item)
- void [itemsAboutToBeRemoved](#) ([GraphicsScene](#) *scene, QList< QGraphicsItem * > &item, QList< [ItemHandle](#) * > &handles, QList< [QUndoCommand](#) * > &)
signals just before items are deleted

- void [itemsRemoved](#) ([GraphicsScene](#) *scene, const QList< [QGraphicsItem](#) * > &item, const QList< [ItemHandle](#) * > &handles)
signals whenever items are deleted
- void [itemsAboutToBeInserted](#) ([GraphicsScene](#) *scene, QList< [QGraphicsItem](#) * > &, QList< [ItemHandle](#) * > &, QList< [QUndoCommand](#) * > &)
signals whenever items are going to be added
- void [itemsInserted](#) ([GraphicsScene](#) *scene, const QList< [QGraphicsItem](#) * > &item, const QList< [ItemHandle](#) * > &handles)
signals whenever items are added
- void [itemsInserted](#) ([NetworkHandle](#) *win, const QList< [ItemHandle](#) * > &)
A convenient signal that is emitted when items are inserted from a [GraphicsScene](#) or [TextEditor](#). Warning: listening to the other itemsInserted signals may cause redundancy.
- void [itemsRemoved](#) ([NetworkHandle](#) *win, const QList< [ItemHandle](#) * > &)
A convenient signal that is emitted when items are removed from a [GraphicsScene](#) or [TextEditor](#). Warning: listening to the other itemsRemoved signals may cause redundancy.
- void [copyItems](#) ([GraphicsScene](#) *scene, QList< [QGraphicsItem](#) * > &, QList< [ItemHandle](#) * > &)
signals just before items are copied
- void [textChanged](#) ([TextEditor](#) *, const QString &, const QString &, const QString &)
some text inside this editor has been changed
- void [lineChanged](#) ([TextEditor](#) *, int, const QString &)
the cursor has moved to a different line
- void [parse](#) ([TextEditor](#) *)
request to parse the text in the current text editor
- void [mouseMoved](#) ([GraphicsScene](#) *scene, [QGraphicsItem](#) *item, [QPointF](#) point, Qt::MouseButton, Qt::KeyboardModifiers modifiers, QList< [QGraphicsItem](#) * > &)
signals whenever mouse moves, and indicates whether it is on top of an item
- void [mouseOnTopOf](#) ([GraphicsScene](#) *scene, [QGraphicsItem](#) *item, [QPointF](#) point, Qt::KeyboardModifiers modifiers, QList< [QGraphicsItem](#) * > &)
signals whenever mouse is on top of an item
- void [sceneRightClick](#) ([GraphicsScene](#) *scene, [QGraphicsItem](#) *item, [QPointF](#) point, Qt::KeyboardModifiers modifiers)
signals whenever right click is made on an item or scene
- void [keyPressed](#) ([GraphicsScene](#) *scene, [QKeyEvent](#) *)
signals whenever a key is pressed
- void [keyReleased](#) ([GraphicsScene](#) *scene, [QKeyEvent](#) *)
signals whenever a key is released

- void [colorChanged](#) ([GraphicsScene](#) *scene, const QList< [QGraphicsItem](#) * > &items)
signals whenever color of items are changed
- void [parentItemChanged](#) ([GraphicsScene](#) *scene, const QList< [QGraphicsItem](#) * > &items, const QList< [QGraphicsItem](#) * > &parents)
signals whenever item parents are changed
- void [itemsRenamed](#) ([NetworkHandle](#) *window, const QList< [ItemHandle](#) * > &items, const QList< [QString](#) > &oldnames, const QList< [QString](#) > &newnames)
signals whenever an item is renamed
- void [handlesChanged](#) ([NetworkHandle](#) *scene, const QList< [QGraphicsItem](#) * > &items, const QList< [ItemHandle](#) * > &old)
signals whenever the handles for graphics items have changed
- void [parentHandleChanged](#) ([NetworkHandle](#) *scene, const QList< [ItemHandle](#) * > &, const QList< [ItemHandle](#) * > &)
signals whenever item parent handle is changed
- void [handleFamilyChanged](#) ([NetworkHandle](#) *network, const QList< [ItemHandle](#) * > &, const QList< [ItemFamily](#) * > &)
signals whenever item handles' families are changed
- void [dataChanged](#) (const QList< [ItemHandle](#) * > &items)
signals whenever some data is changed
- void [escapeSignal](#) (const [QWidget](#) *sender)
signals whenever the current activities need to be stopped
- void [filesLoaded](#) (const QList< [QFileInfo](#) > &files)
signals whenever file(s) are loaded. Each file can be a model or a plugin
- void [itemsDropped](#) ([GraphicsScene](#) *, const [QString](#) &, const [QPointF](#) &)
signal is emitted when some object OTHER than files are dropped on the canvas
- void [saveSettings](#) ()
save initial settings to settingsFileName
- void [loadDefaultPlugins](#) ()
load default plugins
- void [closeEvent](#) ([QCloseEvent](#) *event)
close window event -- asks whether to save file
- virtual void [dropEvent](#) ([QDropEvent](#) *)
drag and drop
- virtual void [dragEnterEvent](#) ([QDragEnterEvent](#) *event)
drag and drop

slots

- void [setUserHome](#) ()
asks user for a new directory to be used as the user home directory (must be writable)
- [GraphicsScene](#) * [newScene](#) ()
create new scene
- [TextEditor](#) * [newTextEditor](#) ()
create new text editor
- void [closeWindow](#) ()
triggered when the close button is clicked. Closes the current window
- void [saveWindow](#) ()
triggered when the save button is clicked. Opens a file dialog and emits the save signal. The main window itself does not implement the save.
- void [saveWindowAs](#) ()
triggered when the save-as button is clicked. Opens a file dialog and emits the save signal. The main window itself does not implement the save.
- void [open](#) ()
triggered when the open button is clicked. Opens a file dialog. Note: the core library just emits a signal, and other tools are responsible for actually opening a file
- void [open](#) (const QString &)
open a file. Note: the core library just emits a signal, and other tools are responsible for actually opening a file The main window does not implement an function for opening a new file
- void [undo](#) ()
calls current scene or text editor's undo
- void [redo](#) ()
calls current scene or text editor's redo
- void [copy](#) ()
calls current scene or text editor's copy
- void [cut](#) ()
calls current scene or text editor's cut
- void [paste](#) ()
calls current scene or text editor's paste
- void [selectAll](#) ()
calls current scene or text editor's selectAll
- void [remove](#) ()
calls current scene or text editor's find

- void [print](#) ()
triggered when the print button is clicked. Calls current scene's print
- void [printToFile](#) ()
triggered when the print-to-file button is clicked. Calls current scene's print on a pdf file
- void [sendEscapeSignal](#) (const QWidget *w=0)
sends a signal to all plugins telling them to exit their current processes.
- void [addParser](#) (TextParser *)
add a new text parser to the list of available parsers. The current text parser can be obtained using TextParser::currentParser();
- void [gridOn](#) ()
change grid mode for current scene to on (>0)
- void [gridOff](#) ()
change grid mode for current scene to off (=0)
- void [setGridSize](#) ()
set grid size for current scene
- void [popOut](#) ()
pop-out the current window
- [ConsoleWindow](#) * [console](#) () const
get the console window
- void [readSettings](#) ()
read initial settings from settingsFileName
- static [MainWindow](#) * [instance](#) ()
gets the global main window
- void [popOut](#) ([NetworkWindow](#) *)
pop-out the given window
- void [popIn](#) ([NetworkWindow](#) *)
pop-in the given window
- void [setCurrentWindow](#) ([NetworkWindow](#) *)
sets the active window
- void [loadFiles](#) (const QList< QFileInfo > &files)
loads files (library files or Network files)
- void [changeConsoleBgColor](#) ()
change console background color
- void [changeConsoleTextColor](#) ()

change console text color

- void [changeConsoleMsgColor](#) ()
change console message text color
- void [changeConsoleErrorMsgColor](#) ()
change console error text color
- virtual void [tabIndexChanged](#) (int)
tab changed
- void [itemsRemovedSlot](#) ([GraphicsScene](#) *scene, const QList< [QGraphicsItem](#) * > &item, const QList< [ItemHandle](#) * > &handles)
signals whenever items are deleted
- void [itemsInsertedSlot](#) ([GraphicsScene](#) *scene, const QList< [QGraphicsItem](#) * > &item, const QList< [ItemHandle](#) * > &handles)
signals whenever items are added
- void [setupFunctionPointersSlot](#) ([QSemaphore](#) *, [QLibrary](#) *)
send signal to other tools so that they can connect functions to signals

6.37.1 Detailed Description

[MainWindow](#) is the parent container for all the other widgets in TinkerCell. The central widget in [MainWindow](#) is a tab widget. Each tab widget can hold a [GraphicsView](#) or a [TextEditor](#). One of the main roles of [MainWindow](#) is to serve as a signal/slot hub for Tools.

6.37.2 Constructor & Destructor Documentation

6.37.2.1 TinkerCell::MainWindow::MainWindow (bool enableScene = true, bool enableText = true, bool views = true)

5-arg (optional) constructor allows disabling of text/graphics modes

Parameters

- bool** enable text-based network construction (default = true)
- bool** enable graphics-based network construction (default = true)
- bool** allow tabbed and windowed view modes (default = true)

6.37.2.2 TinkerCell::MainWindow::~~MainWindow () [virtual]

Destructor: delete all the graphics scenes.

destructor

6.37.3 Member Function Documentation

6.37.3.1 void TinkerCell::MainWindow::addTool (Tool * *tool*)

add a new tool to the list of tools stored in the main window

Parameters

the name of the new tool
the new tool

Returns

void

6.37.3.2 QDockWidget * TinkerCell::MainWindow::addToolWindow (QWidget * *tool*, TOOL_WINDOW_OPTION *option* = *DockWidget*, Qt::DockWidgetArea *initArea* = Qt::RightDockWidgetArea, Qt::DockWidgetAreas *allowedAreas* = Qt::AllDockWidgetAreas, bool *inMenu* = *true*)

Add a new docking window to the main window. The name and icon are obtained using the widget's windowTitle and windowIcon, so be sure to set those before calling this function.

Parameters

*Tool** the new tool
Qt::DockWidgetArea the initial docking area
Qt::DockWidgetAreas the allowed docking areas
bool whether or not to place the docking window in the view menu
bool use a tab widget instead of a dock widget. The widget will not be dockable, but the entire tab widget will be dockable.

Returns

QDockWidget* the new docking widget. TabWidget option is used, the docking widget may be an existing docking widget.

6.37.3.3 void TinkerCell::MainWindow::addToViewMenu (QWidget * *tool*)

place a show/hide action in the view menu for the given widget

Parameters

*QWidget** the new widget

6.37.3.4 void TinkerCell::MainWindow::allowMultipleViewModes (bool *b*) [virtual]

allow or disallow changing between different views

Parameters

bool

6.37.3.5 void Tinkercell::MainWindow::changeConsoleBgColor () [protected, slot]

change console background color

Returns

void

6.37.3.6 void Tinkercell::MainWindow::changeConsoleErrorMsgColor () [protected, slot]

change console error text color

Returns

void

6.37.3.7 void Tinkercell::MainWindow::changeConsoleMsgColor () [protected, slot]

change console message text color

Returns

void

6.37.3.8 void Tinkercell::MainWindow::changeConsoleTextColor () [protected, slot]

change console text color

Returns

void

6.37.3.9 void Tinkercell::MainWindow::closeEvent (QCloseEvent * *event*) [protected]

close window event -- asks whether to save file

Parameters

QCloseEvent * event

Returns

void

6.37.3.10 void Tinkercell::MainWindow::colorChanged (GraphicsScene * *scene*, const QList< QGraphicsItem * > & *items*) [signal]

signals whenever color of items are changed

Parameters

GraphicsScene * scene where the event took place
QList<QGraphicsItem>&* items that changed color

Returns

void

6.37.3.11 void Tinkercell::MainWindow::copyItems (GraphicsScene * *scene*, QList< QGraphicsItem * > &, QList< ItemHandle * > &) [signal]

signals just before items are copied

Parameters

GraphicsScene * scene where the items are going to be copied
QList<QGraphicsItem>&* list of graphics items going to be copied
QList<ItemHandle>&* list of handles going to be copied (does NOT have to be the same number as items removed)

Returns

void

6.37.3.12 NetworkHandle * Tinkercell::MainWindow::currentNetwork () const

gets the current window that is active

Returns

NetworkHandle* current network

6.37.3.13 GraphicsScene * Tinkercell::MainWindow::currentScene () const

gets the current scene that is active

Returns

GraphicsScene* current scene

6.37.3.14 TextEditor * Tinkercell::MainWindow::currentTextEditor () const

gets the text editor that is active

Returns

TextEditor* current editor

6.37.3.15 `NetworkWindow * Tinkercell::MainWindow::currentWindow () const`

gets the current window that is active (each window contains either a scene or editor)

Returns

NetworkWindow* current network window

6.37.3.16 `void Tinkercell::MainWindow::dataChanged (const QList< ItemHandle * > & items) [signal]`

signals whenever some data is changed

Parameters

QList<ItemHandle>&* items handles

Returns

void

6.37.3.17 `void Tinkercell::MainWindow::escapeSignal (const QWidget * sender) [signal]`

signals whenever the current activities need to be stopped

Parameters

*QWidget ** the widget that send the signal

Returns

void

6.37.3.18 `void Tinkercell::MainWindow::filesLoaded (const QList< QFileInfo > & files) [signal]`

signals whenever file(s) are loaded. Each file can be a model or a plugin

Parameters

QList<QFileInfo>& the name(s) of the file(s)

Returns

void

6.37.3.19 `void Tinkercell::MainWindow::funtionPointersToMainThread (QSemaphore *, QLibrary *) [signal]`

used internally by [MainWindow](#) in order to move from a thread to the main thread

Parameters

*QSemaphore** Semaphore that lets the thread run once C API is initialized

QLibrary * the new FuntionToSignal instance

Returns

void

6.37.3.20 void Tinkercell::MainWindow::getItemsFromFile (QList< ItemHandle * > & ,
QList< QGraphicsItem * > & , const QString & *filename*, ItemHandle * *root*)
[signal]

signal sent to a tool so that the tool can get the items inside a file

Parameters

QList<ItemHandle>&* list of items inside the file

QList<QGraphicsItem>&* list of graphics items in the file

QString& file that is selected by user

ItemHandle * optional root parent handle for all the loaded items

Returns

void

6.37.3.21 QPair< QList< ItemHandle * >, QList< QGraphicsItem * > >
Tinkercell::MainWindow::getItemsFromFile (const QString & *filename*, ItemHandle
* *root* = 0)

get the items inside a file. Some tool must implement this function and connect to the getItemsFromFile signal. The Core library does not implement a read file function.

Parameters

QString& file that is selected by user

*ItemHandle** optional parent handle to all the items that will be loaded form file

Returns

QList<ItemHandle*> list of items inside the file
void

6.37.3.22 void Tinkercell::MainWindow::handleFamilyChanged (NetworkHandle * *network*,
const QList< ItemHandle * > & , const QList< ItemFamily * > &) [signal]

signals whenever item handles' families are changed

Parameters

*NetworkHandle** network where the event took place

QList<ItemHandle>&* child items

QList<ItemFamily>&* old families

Returns

void

6.37.3.23 void Tinkercell::MainWindow::handlesChanged (NetworkHandle * *scene*, const QList< QGraphicsItem * > & *items*, const QList< ItemHandle * > & *old*) [signal]

signals whenever the handles for graphics items have changed

Parameters

*GraphicsScene** *scene* where the event took place

QList<GraphicsItem>&* *items* that are affected

QList<ItemHandle>&* *old* handle for each items

Returns

void

6.37.3.24 void Tinkercell::MainWindow::historyChanged (int *i* = 0) [signal]

one of more changed have occurred in the history window of the current scene

Parameters

int *number of changes* (negative = undos, positive = redos)

Returns

void

6.37.3.25 QUndoStack * Tinkercell::MainWindow::historyStack () const

the history stack of the current network.

Returns

QUndoStack* current scene's history stack or null if current network is null

6.37.3.26 QUndoView * Tinkercell::MainWindow::historyWidget ()

the history stack widget of the current window.

Returns

QUndoView* current scene's history stack or null if current network is null

6.37.3.27 void Tinkercell::MainWindow::initializeMenus (bool *enableScene* = *true*, bool *enableText* = *true*)

Initialize the basic menu (save, open, close, exit, etc.).

Returns

void

6.37.3.28 void Tinkercell::MainWindow::itemsAboutToBeInserted (QGraphicsScene * *scene*, QList< QGraphicsItem * > & , QList< ItemHandle * > & , QList< QUndoCommand * > &) [signal]

signals whenever items are going to be added

Parameters

*GraphicsScene** *scene* where the items are added

QList<QGraphicsItem>&* list of new graphics items

QList<ItemHandle>&* list of new handles (does NOT have to be the same number as items)

QList<QUndoCommand>&* list of commands that will be executed right before items are inserted

Returns

void

6.37.3.29 void Tinkercell::MainWindow::itemsAboutToBeMoved (QGraphicsScene * *scene*, QList< QGraphicsItem * > & *item*, QList< QPointF > & *distance*, QList< QUndoCommand * > &) [signal]

signals whenever items are going to be moved (each item is the top-most item)

Parameters

*GraphicsScene** *scene* where the items were moved

QList<QGraphicsItem>&* list of pointers to all moving items

QPointF distance by which items moved

Qt::KeyboardModifiers modifier keys being used when mouse clicked

QList<QUndoCommand>&* list of commands that will be executed right before items are inserted

Returns

void

6.37.3.30 void Tinkercell::MainWindow::itemsAboutToBeRemoved (QGraphicsScene * *scene*, QList< QGraphicsItem * > & *item*, QList< ItemHandle * > & *handles*, QList< QUndoCommand * > &) [signal]

signals just before items are deleted

Parameters

*GraphicsScene** scene where the items are going to be removed

QList<QGraphicsItem>&* list of items going to be removed

QList<ItemHandle>&* list of handles going to be removed (does NOT have to be the same number as items removed)

QList<QUndoCommand>&* list of commands that will be executed right before items are inserted

Returns

void

6.37.3.31 void Tinkercell::MainWindow::itemsDropped (GraphicsScene *, const QString &, const QPointF &) [**signal**]

signal is emitted when some object OTHER than files are dropped on the canvas

Parameters

*GraphicsScene** the scene where objects were dropped

QString the string describing the object that was dropped

QPointF the Scene position where it was dropped

Returns

void

6.37.3.32 void Tinkercell::MainWindow::itemsInserted (GraphicsScene * scene, const QList<QGraphicsItem * > & item, const QList<ItemHandle * > & handles) [**signal**]

signals whenever items are added

Parameters

GraphicsScene * scene where the items were added

QList<QGraphicsItem>&* list of new items

QList<ItemHandle>&* list of new handles (does NOT have to be the same number as items)

Returns

void

6.37.3.33 void Tinkercell::MainWindow::itemsInserted (NetworkHandle * win, const QList<ItemHandle * > &) [**signal**]

A convenient signal that is emitted when items are inserted from a [GraphicsScene](#) or [TextEditor](#). Warning: listening to the other itemsInserted signals may cause redundancy.

Parameters

*NetworkHandle** where the editing happened

QList<TextItem>* new items

6.37.3.34 void Tinkercell::MainWindow::itemsInsertedSlot (GraphicsScene * *scene*, const QList< QGraphicsItem * > & *item*, const QList< ItemHandle * > & *handles*) [protected, slot]

signals whenever items are added

Parameters

[*GraphicsScene*](#) * scene where the items were added

[*QList<QGraphicsItem*>&*](#) list of new items

[*QList<ItemHandle*>&*](#) list of new handles (does NOT have to be the same number as items)

Returns

void

6.37.3.35 void Tinkercell::MainWindow::itemsMoved (GraphicsScene * *scene*, const QList< QGraphicsItem * > & *item*, const QList< QPointF > & *distance*) [signal]

signals whenever items are being moved (each item is the top-most item)

Parameters

[*GraphicsScene*](#) * scene where the items were moved

[*QList<QGraphicsItem*>&*](#) list of pointers to all moving items

[*QPointF*](#) distance by which items moved

[*Qt::KeyboardModifiers*](#) modifier keys being used when mouse clicked

Returns

void

6.37.3.36 void Tinkercell::MainWindow::itemsRemoved (NetworkHandle * *win*, const QList< ItemHandle * > &) [signal]

A convenient signal that is emitted when items are removed from a [*GraphicsScene*](#) or [*TextEditor*](#). Warning: listening to the other itemsRemoved signals may cause redundancy.

Parameters

[*NetworkHandle**](#) where the editing happened

[*ItemHandle**](#) removed items

6.37.3.37 void Tinkercell::MainWindow::itemsRemoved (GraphicsScene * *scene*, const QList< QGraphicsItem * > & *item*, const QList< ItemHandle * > & *handles*) [signal]

signals whenever items are deleted

Parameters

[*GraphicsScene*](#) * scene where the items were removed

QList<QGraphicsItem>&* list of items removed

QList<ItemHandle>&* list of handles removed (does NOT have to be the same number as items removed)

Returns

void

6.37.3.38 void Tinkercell::MainWindow::itemsRemovedSlot (QGraphicsScene * *scene*, const QList< QGraphicsItem * > & *item*, const QList< ItemHandle * > & *handles*)
[protected, slot]

signals whenever items are deleted

Parameters

GraphicsScene * scene where the items were removed

QList<QGraphicsItem>&* list of items removed

QList<ItemHandle>&* list of handles removed (does NOT have to be the same number as items removed)

Returns

void

6.37.3.39 void Tinkercell::MainWindow::itemsRenamed (NetworkHandle * *window*, const QList< ItemHandle * > & *items*, const QList< QString > & *oldnames*, const QList< QString > & *newnames*) [signal]

signals whenever an item is renamed

Parameters

NetworkHandle * window where the event took place

QList<ItemHandle>&* items

QList<QString>& old names

QList<QString>& new names

Returns

void

6.37.3.40 void Tinkercell::MainWindow::itemsSelected (QGraphicsScene * *scene*, const QList< QGraphicsItem * > & *items*, QPointF *point*, Qt::KeyboardModifiers *modifiers*)
[signal]

signals whenever a new item is selected (item can be sub-item, not top-level)

Parameters

GraphicsScene * scene where items are selected

QList<QGraphicsItem>&* list of all selected item pointers

QPointF point where mouse is clicked

Qt::KeyboardModifiers modifier keys being used when mouse clicked

Returns

void

6.37.3.41 void Tinkercell::MainWindow::keyPressed (QGraphicsScene * *scene*, QKeyEvent *)
[signal]

signals whenever a key is pressed

Parameters

GraphicsScene * scene where the event took place

QKeyEvent * key that is pressed

Returns

void

6.37.3.42 void Tinkercell::MainWindow::keyReleased (QGraphicsScene * *scene*, QKeyEvent *)
[signal]

signals whenever a key is released

Parameters

GraphicsScene * scene where the event took place

QKeyEvent * key that is released

Returns

void

6.37.3.43 void Tinkercell::MainWindow::lineChanged (TextEditor *, int, const QString &)
[signal]

the cursor has moved to a different line

Parameters

*TextEditor** editor

int index of the current line

QString current line text

6.37.3.44 void Tinkercell::MainWindow::loadDefaultPlugins () [protected]

load default plugins

Returns

void

6.37.3.45 void Tinkercell::MainWindow::loadDynamicLibrary (const QString & *dllFile*)

Load a new plugin (dll).

Parameters

the complete path of the dll file

Returns

void

**6.37.3.46 void Tinkercell::MainWindow::loadFiles (const QList< QFileInfo > & *files*)
[protected, slot]**

loads files (library files or Network files)

Parameters

QList<QFileInfo>& the name(s) of the file(s)

Returns

void

6.37.3.47 void Tinkercell::MainWindow::loadNetwork (const QString & *filename*) [signal]

signals when user selects a file to open in the current network

Parameters

QString& file that is selected by user

Returns

void

**6.37.3.48 void Tinkercell::MainWindow::mouseDoubleClicked (QGraphicsScene * *scene*,
QPointF *point*, QGraphicsItem *, Qt::MouseButton, Qt::KeyboardModifiers
modifiers) [signal]**

emits event when mouse is double clicked

Parameters

GraphicsScene * scene where the event took place

point where mouse is clicked

modifier keys being used when mouse clicked

Returns

void

6.37.3.49 void Tinkercell::MainWindow::mouseDragged (GraphicsScene * *scene*, QPointF *from*, QPointF *to*, Qt::MouseButton, Qt::KeyboardModifiers *modifiers*) [signal]

signals whenever mouse is dragged from one point to another

Parameters

GraphicsScene * scene where the event took place

QPointF point where mouse is clicked first

QPointF point where mouse is released

Qt::MouseButton button being pressed

Qt::KeyboardModifiers modifier keys being used when mouse clicked

Returns

void

6.37.3.50 void Tinkercell::MainWindow::mouseMoved (GraphicsScene * *scene*, QGraphicsItem * *item*, QPointF *point*, Qt::MouseButton, Qt::KeyboardModifiers *modifiers*, QList< QGraphicsItem * > &) [signal]

signals whenever mouse moves, and indicates whether it is on top of an item

Parameters

GraphicsScene * scene where the event took place

*QGraphicsItem** pointer to item that mouse is on top of

QPointF point where mouse is clicked

Qt::MouseButton button being pressed

Qt::KeyboardModifiers modifier keys being used when mouse clicked

QList<QGraphicsItem>&* list of items that are being moved with the mouse

Returns

void

6.37.3.51 void Tinkercell::MainWindow::mouseOnTopOf (GraphicsScene * *scene*, QGraphicsItem * *item*, QPointF *point*, Qt::KeyboardModifiers *modifiers*, QList<GGraphicsItem * > &) [signal]

signals whenever mouse is on top of an item

Parameters

GraphicsScene * scene where the event took place

*QGraphicsItem** pointer to item that mouse is on top of

QPointF point where mouse is clicked

Qt::KeyboardModifiers modifier keys being used when mouse clicked

QList<QGraphicsItem>&* list of items that are being moved with the mouse

Returns

void

6.37.3.52 void Tinkercell::MainWindow::mousePressed (GraphicsScene * *scene*, QPointF *point*, Qt::MouseButton, Qt::KeyboardModifiers *modifiers*) [signal]

signals whenever an empty node of the screen is clicked

Parameters

GraphicsScene * scene where the event took place

QPointF point where mouse is clicked

Qt::MouseButton which button was pressed

Qt::KeyboardModifiers modifier keys being used when mouse clicked

Returns

void

6.37.3.53 void Tinkercell::MainWindow::mouseReleased (GraphicsScene * *scene*, QPointF *point*, Qt::MouseButton, Qt::KeyboardModifiers *modifiers*) [signal]

signals whenever an empty node of the screen is clicked

Parameters

GraphicsScene * scene where the event took place

QPointF point where mouse is clicked

Qt::MouseButton which button was pressed

Qt::KeyboardModifiers modifier keys being used when mouse clicked

Returns

void

6.37.3.54 void Tinkercell::MainWindow::networkClosed (NetworkHandle *) [signal]

signals after a window is closed

Parameters

NetworkHandle * the window that was closed

Returns

void

6.37.3.55 void Tinkercell::MainWindow::networkClosing (NetworkHandle * , bool *) [signal]

signals when a network is going to close

Parameters

NetworkHandle * the network that is closing

Boolean setting to false will prevent this window from closing

Returns

void

6.37.3.56 void Tinkercell::MainWindow::networkLoaded (NetworkHandle *) [signal]

signals informs that the current network has just loaded a new Network

Parameters

NetworkHandle * the window where network was loaded (usually current scene)

Returns

void

6.37.3.57 void Tinkercell::MainWindow::networkOpened (NetworkHandle *) [signal]

signals whenever the new network is opened

Parameters

*NetworkHandle** the current new window

Returns

void

6.37.3.58 `QList< NetworkHandle * > Tinkercell::MainWindow::networks () const`

gets all the windows in the main window

Returns

`QList<NetworkHandle*>` list of windows

6.37.3.59 `void Tinkercell::MainWindow::networkSaved (NetworkHandle *) [signal]`

signals when a tool has saved the network in a file

Parameters

NetworkHandle * the window where network was loaded (usually current scene)

Returns

void

6.37.3.60 `void Tinkercell::MainWindow::parentHandleChanged (NetworkHandle * scene, const QList< ItemHandle * > &, const QList< ItemHandle * > &) [signal]`

signals whenever item parent handle is changed

Parameters

NetworkHandle * window where the event took place

`QList<ItemHandle*>&` child items

`QList<ItemHandle*>&` old parents

Returns

void

6.37.3.61 `void Tinkercell::MainWindow::parentItemChanged (QGraphicsScene * scene, const QList< QGraphicsItem * > & items, const QList< QGraphicsItem * > & parents) [signal]`

signals whenever item parents are changed

Parameters

GraphicsScene * scene where the event took place

`QList<QGraphicsItem*>&` items

`QList<QGraphicsItem*>&` new parents

Returns

void

6.37.3.62 void Tinkercell::MainWindow::parse (*TextEditor* *) [signal]

request to parse the text in the current text editor

Parameters

*TextEditor** editor

6.37.3.63 void Tinkercell::MainWindow::prepareNetworkForSaving (*NetworkHandle* *, bool *) [signal]

signals when a tool is about to save a network

Parameters

NetworkHandle * the window where Network was loaded (usually current scene)

Returns

void

6.37.3.64 void Tinkercell::MainWindow::print () [slot]

triggered when the print button is clicked. Calls current scene's print
print the current scene

6.37.3.65 void Tinkercell::MainWindow::printToFile () [slot]

triggered when the print-to-file button is clicked. Calls current scene's print on a pdf file
print the current scene

6.37.3.66 void Tinkercell::MainWindow::readSettings () [slot]

read initial settings from settingsFileName

Returns

void

6.37.3.67 void Tinkercell::MainWindow::saveNetwork (const *QString* & *filename*) [signal]

signals when user selects a file to save the current network to

Parameters

QString& file that is selected by user

Returns

void

6.37.3.68 void Tinkercell::MainWindow::saveSettings () [protected]

save initial settings to settingsFileName

Returns

void

6.37.3.69 void Tinkercell::MainWindow::sceneRightClick (GraphicsScene * scene, QGraphicsItem * item, QPointF point, Qt::KeyboardModifiers modifiers) [signal]

signals whenever right click is made on an item or scene

Parameters

GraphicsScene * scene where the event took place

*QGraphicsItem** pointer to item that mouse is clicked on

QPointF point where mouse is clicked

Qt::KeyboardModifiers modifier keys being used when mouse clicked

Returns

void

6.37.3.70 void Tinkercell::MainWindow::setCursor (QCursor cursor)

set the cursor for all windows

Parameters

QCursor cursor

Returns

void

6.37.3.71 void Tinkercell::MainWindow::setupFunctionPointers (QLibrary *) [signal]

signals when a new FuntionToSignal is constructed

Parameters

QLibrary * the new FuntionToSignal instance

Returns

void

6.37.3.72 void Tinkercell::MainWindow::setupFunctionPointersSlot (QSemaphore * *s*,
QLibrary * *library*) [protected, slot]

send signal to other tools so that they can connect functions to signals

Parameters

*QSemaphore** semaphore

QLibrary * the dynamic library instance

Returns

void

6.37.3.73 void Tinkercell::MainWindow::setupNewThread (QSemaphore * *s*, QLibrary * *f*)

This function is usually called from a new thread. This function allows all the plugins to add their functionalities to the C function pointer of the new thread.

Parameters

*QSemaphore** used to wait for all the plugins to initialize the thread

*QLibrary** the library to load

Returns

void

6.37.3.74 void Tinkercell::MainWindow::textChanged (TextEditor *, const QString &, const
QString &, const QString &) [signal]

some text inside this editor has been changed

Parameters

*TextEditor** editor

QString old text (usually a line)

QString new text (usually a line)

6.37.3.75 Tool * Tinkercell::MainWindow::tool (const QString & *s0*) const [virtual]

get a tool

Parameters

QString name of the tool

Returns

Tool*

6.37.3.76 void TinkerCell::MainWindow::toolAboutToBeLoaded (Tool * *tool*, bool * *shouldLoad*) [signal]

a new tool is about to be added. This signal can be used to prevent the tool from being added

Parameters

Tool the tool itself

bool& set this bool to false to prevent the tool from loading

Returns

void

6.37.3.77 void TinkerCell::MainWindow::toolLoaded (Tool * *tool*) [signal]

signals when a new tool (plugin) is loaded

Parameters

*Tool** the new tool

Returns

void

6.37.3.78 QList< Tool * > TinkerCell::MainWindow::tools (const QString & *category* = QString()) const [virtual]

get all tools

Parameters

QString (optional) return only tools in this category, e.g. "plot"

Returns

QList<Tool*>

6.37.3.79 void TinkerCell::MainWindow::windowChanged (NetworkWindow * , NetworkWindow *) [signal]

signals whenever the current window changes

Parameters

*NetworkWindow** the previous windpw

*NetworkWindow** the current new window

Returns

void

The documentation for this class was generated from the following files:

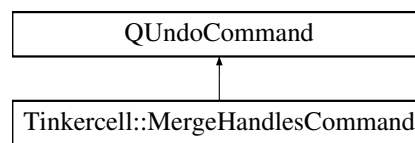
- MainWindow.h
- MainWindow.cpp

6.38 Tinkercell::MergeHandlesCommand Class Reference

this command places all the graphics items inside one handle into the other

```
#include <UndoCommands.h>
```

Inheritance diagram for Tinkercell::MergeHandlesCommand:



Public Member Functions

- **MergeHandlesCommand** (const QString &text, [NetworkHandle](#) *, const QList< [ItemHandle](#) * > &handles)
- void **redo** ()
- void **undo** ()

Public Attributes

- QList< [ItemHandle](#) * > **oldHandles**
- [ItemHandle](#) * **newHandle**

6.38.1 Detailed Description

this command places all the graphics items inside one handle into the other

The documentation for this class was generated from the following files:

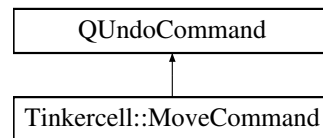
- UndoCommands.h
- UndoCommands.cpp

6.39 Tinkercell::MoveCommand Class Reference

this command performs a move and allows redo/undo of that move

```
#include <UndoCommands.h>
```

Inheritance diagram for Tinkercell::MoveCommand:



Public Member Functions

- **MoveCommand** ([GraphicsScene](#) *scene, QGraphicsItem *item, const QPointF &distance)
constructor
- **MoveCommand** ([GraphicsScene](#) *scene, const QList< QGraphicsItem * > &items, const QPointF &distance)
constructor
- **MoveCommand** ([GraphicsScene](#) *scene, const QList< QGraphicsItem * > &items, const QList< QPointF > &distance)
constructor
- void **redo** ()
redo the change
- void **undo** ()
undo the change

Static Public Member Functions

- static void **refreshAllConnectionIn** (const QList< QGraphicsItem * > &)
refresh all connectors that are attached to any of the items in the list

6.39.1 Detailed Description

this command performs a move and allows redo/undo of that move

6.39.2 Constructor & Destructor Documentation

6.39.2.1 TinkerCell::MoveCommand::MoveCommand ([GraphicsScene](#) * *scene*, QGraphicsItem * *item*, const QPointF & *distance*)

constructor

Parameters

GraphicsScene* *scene* where change happened
QGraphicsItem * *items* that are affected
QPointF& *amount* to move

6.39.2.2 Tinkercell::MoveCommand::MoveCommand (GraphicsScene * *scene*, const QList< QGraphicsItem * > & *items*, const QPointF & *distance*)

constructor

Parameters

scene where change happened
items that are affected
QPointF& amount to move

6.39.2.3 Tinkercell::MoveCommand::MoveCommand (GraphicsScene * *scene*, const QList< QGraphicsItem * > & *items*, const QList< QPointF > & *distance*)

constructor

Parameters

*GraphicsScene** *scene* where change happened
QList<QGraphicsItem>&* *items* that are affected
QPointF& amount to move

6.39.3 Member Function Documentation

6.39.3.1 void Tinkercell::MoveCommand::refreshAllConnectionIn (const QList< QGraphicsItem * > & *moving*) [static]

refresh all connectors that are attached to any of the items in the list

Parameters

items list to check

The documentation for this class was generated from the following files:

- UndoCommands.h
- UndoCommands.cpp

6.40 Tinkercell::MultithreadedSliderWidget Class Reference

This class is used to run specific functions inside a C dynamic library as a separate thread. Uses [CThread](#) to call the C functions.

```
#include <MultithreadedSliderWidget.h>
```

Public Slots

- virtual void [setSliders](#) (const QStringList &options, const QList< double > &minValues, const QList< double > &maxValues)

setup the sliders options and initial values

- virtual void [setVisibleSliders](#) (const QStringList &options)
set the sliders visible
- virtual void [setVisibleSliders](#) (const QString &substring)
set the sliders visible if the slider name has the given string as a substring

Signals

- void [optionsChanged](#) (const QStringList &)
the options in the slider have changed
- void [valuesChanged](#) (const QList< double > &)
the values in the slider have changed

Public Member Functions

- [MultithreadedSliderWidget](#) ([MainWindow](#) *parent, [CThread](#) *thread=0, Qt::Orientation [orientation](#)=Qt::Horizontal)
constructor
- [MultithreadedSliderWidget](#) ([MainWindow](#) *parent, const QString &lib, const QString &function-Name, Qt::Orientation [orientation](#)=Qt::Horizontal)
constructor
- virtual [CThread](#) * [thread](#) () const
the cthread that is run every time the sliders change
- virtual void [setThread](#) ([CThread](#) *)
the cthread that is run every time the sliders change
- virtual void [setDefaultDataTable](#) (const QString &)
This is the data table that will be altered when no appropriate data is available. For example, if one of the sliders is labeled "A" and the default table is set to "bla", then changing the slider for "A" will result in change to "A.bla[0,0]".
- virtual [DataTable](#)< qreal > [data](#) () const
table containing the variables, current values, min and max

Protected Slots

- virtual void [valueChanged](#) ()
whenver the value text change, the function in the C library is called
- virtual void [sliderChanged](#) (int)

whenever the sliders change, the function in the C library is called

- virtual void [minmaxChanged](#) ()
whenever the text change, the function in the C library is called
- virtual void [saveValues](#) ()
copy the values from the slider to the model

Protected Attributes

- [CThread](#) * [cthread](#)
whenever the slides change, cthread->start() is called
- [Qt::Orientation](#) [orientation](#)
orientation of the sliders
- [DataTable](#)< qreal > [values](#)
table storing slider values
- [QList](#)< [QLabel](#) * > [labels](#)
slider labels in same order as sliders list
- [QList](#)< [QSlider](#) * > [sliders](#)
all the sliders
- [QList](#)< [QLineEdit](#) * > [minline](#)
slider min, max, and values in same order as sliders list
- [QList](#)< [QLineEdit](#) * > [maxline](#)
- [QList](#)< [QLineEdit](#) * > [valueline](#)
- [QList](#)< double > [min](#)
slider min and max in same order as sliders list
- [QList](#)< double > [max](#)
- [QVBoxLayout](#) * [slidersLayout](#)
slider layout
- [QHash](#)< [QString](#), [QWidget](#) * > [sliderWidgets](#)
sliders by name
- [MainWindow](#) * [mainWindow](#)
main window
- [QString](#) [defaultDataTable](#)
This is the data table that will be altered when no appropriate data is available. For example, if one of the sliders is labeled "A" and the default table is set to "bla", then changing the slider for "A" will result in change to "A.bla[0,0]".

6.40.1 Detailed Description

This class is used to run specific functions inside a C dynamic library as a separate thread. Uses [CThread](#) to call the C functions.

6.40.2 Constructor & Destructor Documentation

6.40.2.1 TinkerCell::MultithreadedSliderWidget::MultithreadedSliderWidget (MainWindow * parent, CThread * thread = 0, Qt::Orientation orientation = Qt::Horizontal)

constructor

Parameters

QWidget * parent

[CThread](#) * the thread that is already setup with the correct library and function

Qt::Orientation orientation

6.40.2.2 TinkerCell::MultithreadedSliderWidget::MultithreadedSliderWidget (MainWindow * parent, const QString & lib, const QString & functionName, Qt::Orientation orientation = Qt::Horizontal)

constructor

Parameters

QWidget * parent

QString the name of the dynamic library to load

QString name of function in the library with signature void f(Matrix)

Qt::Orientation orientation

6.40.3 Member Function Documentation

6.40.3.1 void TinkerCell::MultithreadedSliderWidget::setSliders (const QStringList & options, const QList< double > & minValues, const QList< double > & maxValues) [virtual, slot]

setup the sliders options and initial values

Parameters

QStringList names for the sliders

QList<double> minimum value for each of the sliders

QList<double> maximum value for each of the sliders

6.40.3.2 void Tinkercell::MultithreadedSliderWidget::setVisibleSliders (const QStringList & options) [virtual, slot]

set the sliders visible

Parameters

QStringList names for the sliders

6.40.3.3 void Tinkercell::MultithreadedSliderWidget::setVisibleSliders (const QString & substring) [virtual, slot]

set the sliders visible if the slider name has the given string as a substring

Parameters

QString substring for the slider names

The documentation for this class was generated from the following files:

- MultithreadedSliderWidget.h
- MultithreadedSliderWidget.cpp

6.41 Tinkercell::NetworkHandle Class Reference

A class that is used to store a network. The network is a collection of Item Handles. The history stack is also a key component of a network. The network can either be represented as text using [TextEditor](#) or visualized with graphical items in the [GraphicsScene](#). Each node and connection are contained in a handle, and each handle can either be represented as text or as graphics. The two main components of [NetworkWindow](#) are the [SymbolsTable](#) and HistoryStack This class provides functions for inserting items, removing items, and changing information inside the model.

```
#include <NetworkHandle.h>
```

Public Slots

slots

update the symbols table that stores all the symbols in the network

- virtual void [updateSymbolsTable](#) ()
updates the symbols table
- virtual void [updateSymbolsTable](#) (int)
updates the symbols table. The int argument is so that this can be connected to the history changed signal
- virtual void [close](#) ()
updates the symbols table. The int argument is so that this can be connected to the history changed signal
- virtual void [undo](#) ()
undo last command

- virtual void [redo](#) ()
redo last command
- virtual void [push](#) (QUndoCommand *)
push a new command into the history stack

Public Member Functions

Constructor and destructor

- [NetworkHandle](#) (MainWindow *)
constructor
- virtual [~NetworkHandle](#) ()
destructor

Get items

get the set of items in the model

- virtual QList< [ItemHandle](#) * > [handles](#) (bool sort=false)
get all the visible items in this network window
- virtual QList< [ItemHandle](#) * > [handlesSortedByFamily](#) () const
get list of all items sorted according to family
- virtual [ItemHandle](#) * [globalHandle](#) ()
the model global item
- virtual QString [annotations](#) () const
all free floating text in the visual diagram
- virtual [GraphicsScene](#) * [currentScene](#) () const
gets the current scene that is active
- virtual [TextEditor](#) * [currentTextEditor](#) () const
gets the text editor that is active
- virtual [NetworkWindow](#) * [currentWindow](#) () const
gets the window that is active
- virtual void [showScene](#) ([GraphicsScene](#) *)
show the window that contains the given scene
- virtual void [showTextEditor](#) ([TextEditor](#) *)
show the window that contains the given text editor
- [ConsoleWindow](#) * [console](#) () const
same as main window's [console\(\)](#)

find item handles and data tables

- `QList< ItemHandle * > findItem (const QString &) const`
get all the items with the given name. Returns a list for non-unique names
- `QList< ItemHandle * > findItem (const QStringList &) const`
get all the items with the given name. returned list may be longer if names are non-unique
- `QList< QPair< ItemHandle *, QString > > findData (const QString &) const`
get all the items and corresponding data table name that contains the given string. if non-unique, returns a list
- `QList< QPair< ItemHandle *, QString > > findData (const QStringList &) const`
get all the items and corresponding data table name that contains the given string. if non-unique, returns a list

create scene or editor

- `virtual void remove (const QString &name, const QList< QGraphicsItem * > &items)`
this command performs a removal and also adds undo command to history window and emits associated signal(s)
- `virtual void remove (const QString &name, const QList< ItemHandle * > &items)`
this command performs a removal and also adds undo command to history window and emits associated signal(s)
- `virtual QList< GraphicsScene * > scenes () const`
get all the graphics scenes used to illustrate this network
- `virtual QList< TextEditor * > editors () const`
get all the text editors used to express this network
- `virtual GraphicsScene * createScene (const QList< QGraphicsItem * > &insertItems=QList< QGraphicsItem * >())`
create a new scene for this network
- `virtual GraphicsScene * createScene (ItemHandle *, const QRectF &boundingRect=QRectF())`
create a new scene that gets all the items inside the given item handle.
- `virtual TextEditor * createTextEditor (const QString &text=QString())`
create a new text editor for this network
- `virtual void setWindowTitle (const QString &)`
set all the title for each window representing this network
- `virtual QString windowTitle () const`
get the title for current window representing this network
- `virtual bool parseMath (QString &, QStringList &)`
checks whether a string is a correct formula.
- `virtual QString makeUnique (const QString &, const QStringList &doNotUseNames=QStringList()) const`
checks whether the given string names a unique item or data entry
- `virtual QString makeUnique (ItemHandle *handle, const QStringList &doNotUseNames=QStringList()) const`

checks whether the given handle's name is unique and returns a new name. Note that this can be different from `makeUnique` for strings, because this function will check if an existing name belongs to the given handle, in which case no change is needed.

- virtual QStringList `makeUnique` (const QStringList &, const QStringList &doNotUseNames=QStringList()) const
checks whether the given string names a unique item or data entry

rename items

These functions automatically perform history updates and send appropriate signals, which will inform the other tools that an insertion or deletion has taken place.

- virtual void `rename` (const QString &oldname, const QString &new_name)
rename item and also adds undo command to history window and emits associated signal(s)
- virtual void `rename` (ItemHandle *item, const QString &new_name)
rename an item and also adds undo command to history window and emits associated signal(s)
- virtual void `rename` (const QList< ItemHandle * > &items, const QList< QString > &new_names)
rename items and also adds undo command to history window and emits associated signal(s)

change parents of items

These functions automatically perform history updates and send appropriate signals, which will inform the other tools that an insertion or deletion has taken place.

- virtual void `setParentHandle` (const QList< ItemHandle * > &handles, const QList< ItemHandle * > &parentHandles)
change parent handles and also adds undo command to history window and emits associated signal(s)
- virtual void `setParentHandle` (ItemHandle *child, ItemHandle *parent)
change parent handle and also adds undo command to history window and emits associated signal(s)
- virtual void `setParentHandle` (const QList< ItemHandle * > children, ItemHandle *parent)
change parent for handles and also adds undo command to history window and emits associated signal(s)
- virtual void `setHandleFamily` (const QList< ItemHandle * > &handles, const QList< ItemFamily * > &newfamilies)
change handles families and also adds undo command to history window and emits associated signal(s)
- virtual void `setHandleFamily` (ItemHandle *handle, ItemFamily *newfamily)
change handle and also adds undo command to history window and emits associated signal(s)
- virtual void `setHandleFamily` (const QList< ItemHandle * > handles, ItemFamily *newfamily)
change family for handles and also adds undo command to history window and emits associated signal(s)

change data in one or more items

These functions automatically perform history updates and send appropriate signals, which will inform the other tools that an insertion or deletion has taken place.

- virtual void `changeData` (const QString &name, `ItemHandle` *handle, const QString &hashstring, const `NumericalDataTable` *newdata)
change numerical data table and also adds undo command to history window and emits associated signal(s)
- virtual void `changeData` (const QString &name, const QList< `ItemHandle` * > &handles, const QList< QString > &hashstring, const QList< `NumericalDataTable` * > &newdata)
change a list of numerical data tables and also adds undo command to history window and emits associated signal(s)
- virtual void `changeData` (const QString &name, const QList< `ItemHandle` * > &handles, const QString &hashstring, const QList< `NumericalDataTable` * > &newdata)
change a list of numerical data tables and also adds undo command to history window and emits associated signal(s)
- virtual void `changeData` (const QString &name, `ItemHandle` *handle, const QString &hashstring, const `TextDataTable` *newdata)
change text data table and also adds undo command to history window and emits associated signal(s)
- virtual void `changeData` (const QString &name, const QList< `ItemHandle` * > &handles, const QList< QString > &hashstring, const QList< `TextDataTable` * > &newdata)
change a list of text data tables and also adds undo command to history window and emits associated signal(s)
- virtual void `changeData` (const QString &name, const QList< `ItemHandle` * > &handles, const QString &hashstring, const QList< `TextDataTable` * > &newdata)
change a list of text data tables and also adds undo command to history window and emits associated signal(s)
- virtual void `changeData` (const QString &name, `ItemHandle` *handle, const QString &hashstring, const `NumericalDataTable` *newdata1, const `TextDataTable` *newdata2)
change two types of data tables and also adds undo command to history window and emits associated signal(s)
- virtual void `changeData` (const QString &name, const QList< `ItemHandle` * > &handles, const QList< QString > &hashstring, const QList< `NumericalDataTable` * > &newdata1, const QList< `TextDataTable` * > &newdata2)
change a list of two types of data tables and also adds undo command to history window and emits associated signal(s)
- virtual void `changeData` (const QString &name, const QList< `ItemHandle` * > &handles, const QString &hashstring, const QList< `NumericalDataTable` * > &newdata1, const QList< `TextDataTable` * > &newdata2)
change a list of two types of data tables and also adds undo command to history window and emits associated signal(s)
- virtual void `changeData` (const QString &name, const QList< `ItemHandle` * > &handles, const QList< `NumericalDataTable` * > &olddata1, const QList< `NumericalDataTable` * > &newdata1)

change a list of two types of data tables and also adds undo command to history window and emits associated signal(s)
- virtual void `changeData` (const QString &name, const QList< `ItemHandle` * > &handles, const QList< `TextDataTable` * > &olddata2, const QList< `TextDataTable` * > &newdata2)
change a list of two types of data tables and also adds undo command to history window and emits associated signal(s)

- virtual void [changeData](#) (const QString &name, const QList< [ItemHandle](#) * > &handles, const QList< [NumericalDataTable](#) * > &olddata1, const QList< [NumericalDataTable](#) * > &newdata1, const QList< [TextDataTable](#) * > &olddata2, const QList< [TextDataTable](#) * > &newdata2)
change a list of two types of data tables and also adds undo command to history window and emits associated signal(s)
- virtual void [changeData](#) (const QString &name, const QList< [ItemHandle](#) * > &handles, [NumericalDataTable](#) *olddata1, const [NumericalDataTable](#) *newdata1, [TextDataTable](#) *olddata2, const [TextDataTable](#) *newdata2)
change a two types of data tables and also adds undo command to history window and emits associated signal(s)
- virtual void [changeData](#) (const QString &name, const QList< [ItemHandle](#) * > &handles, [NumericalDataTable](#) *olddata1, const [NumericalDataTable](#) *newdata1)
change a data table and also adds undo command to history window and emits associated signal(s)
- virtual void [changeData](#) (const QString &name, const QList< [ItemHandle](#) * > &handles, [TextDataTable](#) *olddata1, const [TextDataTable](#) *newdata1)
change a data table and also adds undo command to history window and emits associated signal(s)
- virtual void [assignHandles](#) (const QList< QGraphicsItem * > &items, [ItemHandle](#) *newHandle)
assign the handle for one or more items
- virtual void [mergeHandles](#) (const QList< [ItemHandle](#) * > &handles)
merge the graphics items and children of two or more handles
- virtual void [setModelValues](#) (const QStringList &names, const QList< double > &values, int column=0, const QString &defaultDataTable=QString())
assign the values for the given strings. if data table has multiple columns, provide the column number in the argument
- virtual void [setModelValues](#) (const QStringList &names, const QStringList &values, int column=0, const QString &defaultDataTable=QString())
assign the values for the given strings. if data table has multiple columns, provide the column number in the argument
- virtual void [setModelValues](#) (const [NumericalDataTable](#) &newvalues, const QString &defaultDataTable=QString())
assign the values for the given strings. if data table has multiple columns, provide the column number in the argument
- virtual void [setModelValues](#) (const [TextDataTable](#) &newvalues, const QString &defaultDataTable=QString())
assign the values for the given strings. if data table has multiple columns, provide the column number in the argument

Public Attributes

- QUndoStack [history](#)
the undo stack

- [SymbolsTable](#) `symbolsTable`

holds a hash of all items and data in this scene.

signals

- class **GraphicsView**
- class **GraphicsScene**
- class **TextEditor**
- class **MainWindow**
- class **NetworkWindow**
- class **SymbolsTable**
- void `itemsRenamed` ([NetworkHandle](#) *network, const QList< [ItemHandle](#) * > &items, const QList< QString > &oldnames, const QList< QString > &newnames)
signals whenever an item is renamed
- void `parentHandleChanged` ([NetworkHandle](#) *network, const QList< [ItemHandle](#) * > &, const QList< [ItemHandle](#) * > &)
signals whenever item parent handle is changed
- void `handleFamilyChanged` ([NetworkHandle](#) *network, const QList< [ItemHandle](#) * > &, const QList< [ItemFamily](#) * > &)
signals whenever item handles' families are changed
- void `dataChanged` (const QList< [ItemHandle](#) * > &items)
signals whenever some data is changed
- void `handlesChanged` ([NetworkHandle](#) *network, const QList< QGraphicsItem * > &items, const QList< [ItemHandle](#) * > &old)
signals whenever the handles for graphics items have changed
- void `historyChanged` (int i=0)
one of more changed have occurred in the history window of the current scene

6.41.1 Detailed Description

A class that is used to store a network. The network is a collection of Item Handles. The history stack is also a key component of a network. The network can either be represented as text using [TextEditor](#) or visualized with graphical items in the [GraphicsScene](#). Each node and connection are contained in a handle, and each handle can either be represented as text or as graphics. The two main components of [NetworkWindow](#) are the [SymbolsTable](#) and [HistoryStack](#). This class provides functions for inserting items, removing items, and changing information inside the model.

6.41.2 Member Function Documentation

6.41.2.1 QString Tinkercell::NetworkHandle::annotations () const **[virtual]**

all free floating text in the visual diagram

Returns

QString

6.41.2.2 void Tinkercell::NetworkHandle::changeData (const QString & *name*, ItemHandle * *handle*, const QString & *hashstring*, const NumericalDataTable * *newdata*) [virtual]

change numerical data table and also adds undo command to history window and emits associated signal(s)
change numerical data table

6.41.2.3 void Tinkercell::NetworkHandle::changeData (const QString & *name*, const QList< ItemHandle * > & *handles*, const QList< QString > & *hashstring*, const QList< NumericalDataTable * > & *newdata*) [virtual]

change a list of numerical data tables and also adds undo command to history window and emits associated signal(s)
change a list of numerical data tables

6.41.2.4 void Tinkercell::NetworkHandle::changeData (const QString & *name*, const QList< ItemHandle * > & *handles*, const QString & *hashstring*, const QList< NumericalDataTable * > & *newdata*) [virtual]

change a list of numerical data tables and also adds undo command to history window and emits associated signal(s)
change a list of numerical data tables

6.41.2.5 void Tinkercell::NetworkHandle::changeData (const QString & *name*, ItemHandle * *handle*, const QString & *hashstring*, const TextDataTable * *newdata*) [virtual]

change text data table and also adds undo command to history window and emits associated signal(s)
change text data table

6.41.2.6 void Tinkercell::NetworkHandle::changeData (const QString & *name*, const QList< ItemHandle * > & *handles*, const QList< QString > & *hashstring*, const QList< TextDataTable * > & *newdata*) [virtual]

change a list of text data tables and also adds undo command to history window and emits associated signal(s)
change a list of text data tables

6.41.2.7 void Tinkercell::NetworkHandle::changeData (const QString & *name*, const QList< ItemHandle * > & *handles*, const QString & *hashstring*, const QList< TextDataTable * > & *newdata*) [virtual]

change a list of text data tables and also adds undo command to history window and emits associated signal(s)

change a list of text data tables

6.41.2.8 void TinkerCell::NetworkHandle::changeData (const QString & *name*, ItemHandle * *handle*, const QString & *hashstring*, const NumericalDataTable * *newdata1*, const TextDataTable * *newdata2*) [virtual]

change two types of data tables and also adds undo command to history window and emits associated signal(s)

change two types of data tables

6.41.2.9 void TinkerCell::NetworkHandle::changeData (const QString & *name*, const QList< ItemHandle * > & *handles*, const QList< QString > & *hashstring*, const QList< NumericalDataTable * > & *newdata1*, const QList< TextDataTable * > & *newdata2*) [virtual]

change a list of two types of data tables and also adds undo command to history window and emits associated signal(s)

change a list of two types of data tables

6.41.2.10 void TinkerCell::NetworkHandle::changeData (const QString & *name*, const QList< ItemHandle * > & *handles*, const QString & *hashstring*, const QList< NumericalDataTable * > & *newdata1*, const QList< TextDataTable * > & *newdata2*) [virtual]

change a list of two types of data tables and also adds undo command to history window and emits associated signal(s)

change a list of two types of data tables

6.41.2.11 QGraphicsScene * TinkerCell::NetworkHandle::createScene (const QList< QGraphicsItem * > & *insertItems* = QList<QGraphicsItem*> ()) [virtual]

create a new scene for this network

Parameters

QList<QGraphicsItem>* items to initialize the network with

Returns

GraficsScene* the new scene

6.41.2.12 QGraphicsScene * TinkerCell::NetworkHandle::createScene (ItemHandle * *item*, const QRectF & *boundingRect* = QRectF ()) [virtual]

create a new scene that gets all the items inside the given item handle.

Parameters

ItemHandle *

QRectF only include the graphicss items

Returns

GraphicsScene* the new scene

6.41.2.13 `TextEditor * Tinkercell::NetworkHandle::createTextEditor (const QString & text = QString()) [virtual]`

create a new text editor for this network

Parameters

QString (optional) initial script

Returns

TextEditor* the new scene

6.41.2.14 `GraphicsScene * Tinkercell::NetworkHandle::currentScene () const [virtual]`

gets the current scene that is active

Returns

GraphicsScene* current scene

6.41.2.15 `TextEditor * Tinkercell::NetworkHandle::currentTextEditor () const [virtual]`

gets the text editor that is active

Returns

TextEditor* current editor

6.41.2.16 `NetworkWindow * Tinkercell::NetworkHandle::currentWindow () const [virtual]`

gets the window that is active

Returns

NetworkWindow* current window

6.41.2.17 `void Tinkercell::NetworkHandle::dataChanged (const QList< ItemHandle * > & items) [signal]`

signals whenever some data is changed

Parameters

QList<ItemHandle>&* items handles

Returns

void

6.41.2.18 `QList< TextEditor * > Tinkercell::NetworkHandle::editors () const [virtual]`

get all the text editors used to express this network

Returns

QList<TextEditor>*

6.41.2.19 `QList< QPair< ItemHandle *, QString > > Tinkercell::NetworkHandle::findData (const QString & s) const`

get all the items and corresponding data table name that contains the given string. if non-unique, returns a list

Parameters

QString

Returns

QPair<ItemHandle,QString>*

6.41.2.20 `QList< QPair< ItemHandle *, QString > > Tinkercell::NetworkHandle::findData (const QStringList & list) const`

get all the items and corresponding data table name that contains the given string. if non-unique, returns a list

Parameters

QString

Returns

QPair<ItemHandle,QString>*

6.41.2.21 `QList< ItemHandle * > TinkerCell::NetworkHandle::findItem (const QString & s) const`

get all the items with the given name. Returns a list for non-unique names

Parameters

QString

Returns

`QList<ItemHandle*>`

6.41.2.22 `QList< ItemHandle * > TinkerCell::NetworkHandle::findItem (const QStringList & list) const`

get all the items with the given name. returned list may be longer if names are non-unique

Parameters

QStringList

Returns

`QList<ItemHandle*>`

6.41.2.23 `void TinkerCell::NetworkHandle::handleFamilyChanged (NetworkHandle * network, const QList< ItemHandle * > &, const QList< ItemFamily * > &) [signal]`

signals whenever item handles' families are changed

Parameters

*NetworkHandle** network where the event took place

QList<ItemHandle>&* child items

QList<ItemFamily>&* old families

Returns

void

6.41.2.24 `QList< ItemHandle * > TinkerCell::NetworkHandle::handles (bool sort = false) [virtual]`

get all the visible items in this network window

Parameters

bool sort handles by full name (default = false)

6.41.2.25 void Tinkercell::NetworkHandle::handlesChanged (NetworkHandle * *network*,
const QList< QGraphicsItem * > & *items*, const QList< ItemHandle * > & *old*)
[signal]

signals whenever the handles for graphics items have changed

Parameters

*NetworkHandle** *network* where the event took place

QList<GraphicsItem>&* *items* that are affected

QList<ItemHandle>&* *old* handle for each items

Returns

void

6.41.2.26 void Tinkercell::NetworkHandle::historyChanged (int *i* = 0) [signal]

one of more changed have occurred in the history window of the current scene

Parameters

int *i* number of changes (negative = undos, positive = redos)

Returns

void

6.41.2.27 void Tinkercell::NetworkHandle::itemsRenamed (NetworkHandle * *network*, const
QList< ItemHandle * > & *items*, const QList< QString > & *oldnames*, const QList<
QString > & *newnames*) [signal]

signals whenever an item is renamed

Parameters

*NetworkHandle** *network* where the event took place

QList<ItemHandle>&* *items*

QList<QString>& *old* names

QList<QString>& *new* names

Returns

void

6.41.2.28 QString Tinkercell::NetworkHandle::makeUnique (const QString & *str*, const
QStringList & *doNotUseNames* = *QStringList()*) const [virtual]

checks whether the given string names a unique item or data entry

Parameters*QString* target string*QStringList* any other names that should be disallowed (optional)**Returns**

QString new string

6.41.2.29 `QString Tinkercell::NetworkHandle::makeUnique (ItemHandle * handle, const QStringList & doNotUseNames = QStringList()) const` **[virtual]**

checks whether the given handle's name is unique and returns a new name. Note that this can be different from makeUnique for strings, because this function will check if an existing name belongs to the given handle, in which case no change is needed.

Parameters*ItemHandle* * handle*QStringList* any other names that should be disallowed (optional)**Returns**

QString new string

6.41.2.30 `QStringList Tinkercell::NetworkHandle::makeUnique (const QStringList & oldnames, const QStringList & doNotUseNames = QStringList()) const` **[virtual]**

checks whether the given string names a unique item or data entry

Parameters*QStringList* target strings**Returns**

QStringList new strings

6.41.2.31 `void Tinkercell::NetworkHandle::parentHandleChanged (NetworkHandle * network, const QList< ItemHandle * > &, const QList< ItemHandle * > &)` **[signal]**

signals whenever item parent handle is changed

Parameters*NetworkHandle** network where the event took place*QList*<*ItemHandle**>& child items*QList*<*ItemHandle**>& old parents**Returns**

void

6.41.2.32 `bool Tinkercell::NetworkHandle::parseMath (QString & s, QStringList & newvars) [virtual]`

checks whether a string is a correct formula.

Parameters

QString target string (also the output)

QStringList returns any new variables not found in this network

Returns

Boolean whether or not the string is valid

6.41.2.33 `QList< GraphicsScene * > Tinkercell::NetworkHandle::scenes () const [virtual]`

get all the graphics scenes used to illustrate this network

Returns

`QList<GraphicsScene*>`

6.41.2.34 `void Tinkercell::NetworkHandle::setModelValues (const QStringList & names, const QStringList & values, int column = 0, const QString & defaultDataTable = QString()) [virtual]`

assign the values for the given strings. if data table has multiple columns, provide the column number in the argument

Parameters

QStringList names of variables

QStringList values

int column number (default=0)

6.41.2.35 `void Tinkercell::NetworkHandle::setModelValues (const NumericalDataTable & newvalues, const QString & defaultDataTable = QString()) [virtual]`

assign the values for the given strings. if data table has multiple columns, provide the column number in the argument

Parameters

NumericalDataTable names (rows) and values

int column number (default=0)

6.41.2.36 void TinkerCell::NetworkHandle::setModelValues (const TextDataTable & *newvalues*, const QString & *defaultDataTable* = *QString()*) [virtual]

assign the values for the given strings. if data table has multiple columns, provide the column number in the argument

Parameters

NumericalDataTable names (rows) and values

int column number (default=0)

6.41.2.37 void TinkerCell::NetworkHandle::setModelValues (const QStringList & *names*, const QList< double > & *values*, int *column* = 0, const QString & *defaultDataTable* = *QString()*) [virtual]

assign the values for the given strings. if data table has multiple columns, provide the column number in the argument

Parameters

QStringList names of variables

QList<double> values

int column number (default=0)

6.41.2.38 void TinkerCell::NetworkHandle::setWindowTitle (const QString & *title*) [virtual]

set all the title for each window representing this network

Parameters

QString

6.41.2.39 void TinkerCell::NetworkHandle::showScene (QGraphicsScene * *scene*) [virtual]

show the window that contains the given scene

Returns

[GraphicsScene](#) * scene

6.41.2.40 void TinkerCell::NetworkHandle::showTextEditor (TextEditor * *editor*) [virtual]

show the window that contains the given text editor

Returns

[TextEditor](#) * text editor

6.41.2.41 void Tinkercell::NetworkHandle::updateSymbolsTable () [virtual, slot]

updates the symbols table

update symbols table

6.41.2.42 void Tinkercell::NetworkHandle::updateSymbolsTable (int i) [virtual, slot]

updates the symbols table. The int argument is so that this can be connected to the history changed signal

update symbols table

6.41.2.43 QString Tinkercell::NetworkHandle::windowTitle () const [virtual]

get the title for current window representing this network

Returns

QString

6.41.3 Member Data Documentation**6.41.3.1 SymbolsTable Tinkercell::NetworkHandle::symbolsTable**

holds a hash of all items and data in this scene.

See also

[SymbolsTable](#)

The documentation for this class was generated from the following files:

- NetworkHandle.h
- NetworkHandle.cpp

6.42 Tinkercell::NetworkWindow Class Reference**Public Slots**

- virtual void [popOut](#) ()
calls main window's popOut
- virtual void [popIn](#) ()
calls main window's popIn
- virtual void [setFileName](#) (const QString &)
set file name and window title
- virtual void [setWindowTitle](#) (const QString &)
set window title

Signals

- void [networkClosing](#) ([NetworkHandle](#) *, bool *)
signals when a window is going to close
- void [networkClosed](#) ([NetworkHandle](#) *)
signals after a window is closed

Public Member Functions

- virtual [GraphicsScene](#) * [newScene](#) ()
replace the current text editor or scene with a new scene
- virtual [TextEditor](#) * [newTextEditor](#) ()
replace the current text editor or scene with a new text editor

Public Attributes

- [NetworkHandle](#) * [network](#)
the network displayed in this window
- [ItemHandle](#) * [handle](#)
this pointer will be non-zero if an [ItemHandle](#) is associated with this window
- [GraphicsScene](#) * [scene](#)
the scene inside this window. Either the scene or the editor must be 0
- [TextEditor](#) * [editor](#)
the editor inside this window. Either the scene or the editor must be 0

Protected Member Functions

- virtual void [closeEvent](#) (QCloseEvent *event)
close event sends signal to all tools asking for confirmation before closing
- virtual void [focusInEvent](#) (QFocusEvent *)
focus received changes the main windows current network pointer
- virtual void [resizeEvent](#) (QResizeEvent *event)
resize event checks if the window has been minimized and calls [popIn](#) instead of minimizing
- virtual void [setAsCurrentWindow](#) ()
calls main window's [setAsCurrentWindow](#)
- virtual void [changeEvent](#) (QEvent *event)
calls [popIn](#) when minimized

- virtual void `connectToMainWindow ()`
make all the main window connections
- `NetworkWindow (NetworkHandle *network, GraphicsScene *scene)`
constructor with scene
- `NetworkWindow (NetworkHandle *network, TextEditor *editor)`
constructor with text editor
- virtual `~NetworkWindow ()`
destructor

Protected Attributes

- QString `filename`
filename associated with this window

Friends

- class `MainWindow`
- class `GraphicsScene`
- class `GraphicsView`
- class `TextEditor`
- class `NetworkHandle`
- class `SymbolsTable`

6.42.1 Member Function Documentation

6.42.1.1 void Tinkercell::NetworkWindow::changeEvent (QEvent * event) [protected, virtual]

calls popIn when minimized

Returns

void

6.42.1.2 void Tinkercell::NetworkWindow::closeEvent (QCloseEvent * event) [protected, virtual]

close event sends signal to all tools asking for confirmation before closing

Parameters

QCloseEvent * event

Returns

void

6.42.1.3 void Tinkercell::NetworkWindow::focusInEvent (QFocusEvent *) [protected, virtual]

focus received changes the main windows current network pointer

Parameters

*QFocusEvent**

Returns

void

6.42.1.4 void Tinkercell::NetworkWindow::networkClosed (NetworkHandle *) [signal]

signals after a window is closed

Parameters

NetworkWindow * the window that was closed

Returns

void

6.42.1.5 void Tinkercell::NetworkWindow::networkClosing (NetworkHandle *, bool *) [signal]

signals when a window is going to close

Parameters

NetworkWindow * the window that is closing

Boolean setting to false will prevent this window from closing

Returns

void

6.42.1.6 GraphicsScene * Tinkercell::NetworkWindow::newScene () [virtual]

replace the current text editor or scene with a new scene

Returns

GraphicsScene * scene

6.42.1.7 TextEditor * Tinkercell::NetworkWindow::newTextEditor () [virtual]

replace the current text editor or scene with a new text editor

Returns

GraphicsScene * scene

6.42.1.8 void Tinkercell::NetworkWindow::popIn () [virtual, slot]

calls main window's popIn

Returns

void

6.42.1.9 void Tinkercell::NetworkWindow::popOut () [virtual, slot]

calls main window's popOut

Returns

void

**6.42.1.10 void Tinkercell::NetworkWindow::resizeEvent (QResizeEvent * event)
[protected, virtual]**

resize event checks if the window has been minimized and calls popIn instead of minimizing

Parameters

*QResizeEvent**

Returns

void

6.42.1.11 void Tinkercell::NetworkWindow::setAsCurrentWindow () [protected, virtual]

calls main window's setAsCurrentWindow

Returns

void

6.42.1.12 void Tinkercell::NetworkWindow::setFileName (const QString & text) [virtual, slot]

set file name and window title

Returns

void

6.42.1.13 void TinkerCell::NetworkWindow::setWindowTitle (const QString & text) [virtual, slot]

set window title

Returns

void

The documentation for this class was generated from the following files:

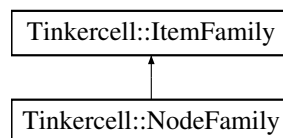
- NetworkWindow.h
- NetworkWindow.cpp

6.43 TinkerCell::NodeFamily Class Reference

This class defines the family of a node. Inherits from [ItemFamily](#). It contains a list of NodeGraphicsItems that is the default for this family of nodes.

```
#include <ItemFamily.h>
```

Inheritance diagram for TinkerCell::NodeFamily:



Public Member Functions

- virtual [ItemFamily](#) * [parent](#) () const
get the parent for this family. If there are more than one parents, returns the first
- virtual QList< [ItemFamily](#) * > [parents](#) () const
get all the parents for this family.
- virtual QList< [ItemFamily](#) * > [children](#) () const
get all the families that make up this family.
- virtual void [setParent](#) ([NodeFamily](#) *)
set parent family
- virtual ~[NodeFamily](#) ()
destructor.
- [NodeFamily](#) (const QString &name=QString())
constructor.
- virtual bool [isA](#) (const QString &) const

indicates whether or not the given string is the name of this family or any of its parent families

- virtual bool `isA` (const `ItemFamily *`) const

indicates whether or not the given family is the name of this family or any of its parent families

Static Public Member Functions

- static `NodeFamily * cast` (`ItemFamily *`)

cast to connection family

Protected Member Functions

- virtual bool `isA` (int) const

indicates whether or not the given ID is this family or any of its parent families

Protected Attributes

- `QList< NodeFamily * > parentFamilies`

all the parents

- `QList< NodeFamily * > childFamilies`

all the families that are under this family

Friends

- class `ConnectionFamily`

6.43.1 Detailed Description

This class defines the family of a node. Inherits from `ItemFamily`. It contains a list of `NodeGraphicsItems` that is the default for this family of nodes.

6.43.2 Constructor & Destructor Documentation

6.43.2.1 `Tinkercell::NodeFamily::NodeFamily (const QString & name = QString())`

constructor.

Parameters

QString name

6.43.3 Member Function Documentation

6.43.3.1 bool TinkerCell::NodeFamily::isA (int *id*) const [protected, virtual]

indicates whether or not the given ID is this family or any of its parent families

indicates whether or not the given string is the name of this family or any of its parent families

Reimplemented from [TinkerCell::ItemFamily](#).

The documentation for this class was generated from the following files:

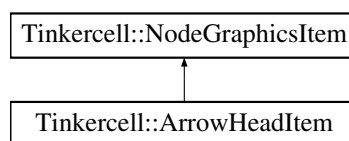
- ItemFamily.h
- ItemFamily.cpp

6.44 TinkerCell::NodeGraphicsItem Class Reference

A simple figure made from one or more polygons. The class can be represented in an XML file.

```
#include <NodeGraphicsItem.h>
```

Inheritance diagram for TinkerCell::NodeGraphicsItem:



Classes

- class [ControlPoint](#)
a control point with a pointer to a [NodeGraphicsItem](#)
- class [Shape](#)
A closed polygon path made from arcs, lines, and beziers.

Public Types

- enum [ShapeType](#) { **arc**, **line**, **bezier**, **rectangle** }
arc, line, or bezier
- enum { **Type** = UserType + 4 }
for enabling dynamic_cast

Public Member Functions

- virtual [ItemHandle](#) * **handle** () const

get the handle of this node

- virtual void [setHandle](#) ([ItemHandle](#) *)

set the handle of this node

- [NodeGraphicsItem](#) ([QGraphicsItem](#) *parent=0)
- [NodeGraphicsItem](#) (const [QString](#) &filename, [QGraphicsItem](#) *parent=0)
- [NodeGraphicsItem](#) (const [NodeGraphicsItem](#) ©)
- virtual [NodeGraphicsItem](#) & [operator=](#) (const [NodeGraphicsItem](#) ©)
- virtual [NodeGraphicsItem](#) * [clone](#) () const

make a copy of this node item

- virtual void [paint](#) ([QPainter](#) *painter, const [QStyleOptionGraphicsItem](#) *option=new [QStyleOptionGraphicsItem](#)(), [QWidget](#) *widget=0)

paint method. Call's parent's paint after setting antialiasing to true

- bool [isValid](#) () const

checks that this is a valid drawable

- virtual void [addControlPoint](#) ([ControlPoint](#) *control)

add a new control point

- virtual void [addShape](#) ([Shape](#) *shape)

add a shape to the set of shapes

- virtual void [removeControlPoint](#) ([ControlPoint](#) *control)

remove a control point

- virtual void [removeShape](#) ([Shape](#) *shape)

add a shape to the set of shapes

- virtual void [setBrush](#) (const [QBrush](#) &newBrush)

change fill color of all shapes

- virtual void [setAlpha](#) (int value)

change alpha value for brush and pen of all shapes

- virtual void [setPen](#) (const [QPen](#) &newPen)

change outline color of all shapes

- virtual void [resetBrush](#) ()

change fill color of all shapes to the default brush

- virtual void [resetPen](#) ()

change outline color of all shapes to default pen

- virtual void [resetToDefaults](#) ()

change color, transformation, and size to defaults

- virtual [QPolygonF](#) [polygon](#) () const

gets a polygon that represents this graphicsItem

- virtual QPainterPath [shape](#) () const
gets a path that represents this graphicsItem
- virtual void [refresh](#) ()
Updates the graphicsItem by re-initializing the vector of shapes Precondition: shapes.size > 1 Postcondition: NA.
- virtual void [normalize](#) ()
*normalizes a node graphics item so that its center is 0,0 and width*height is 10*
- virtual void [clear](#) ()
Clear all shapes and control points.
- virtual QRectF [boundingRect](#) () const
bounding rect
- virtual ~NodeGraphicsItem ()
Destructor: deletes all shapes and control points.
- virtual QList< [Tinkercell::ControlPoint](#) * > [allControlPoints](#) () const
all the control points that are used in this figure
- virtual void [adjustBoundaryControlPoints](#) ()
reset of control points that control the bounding box of this figure
- virtual void [adjustToBoundaryControlPoints](#) ()
set boundary to match control points that control the bounding box of this figure
- virtual void [setBoundingRect](#) (const QPointF &, const QPointF &)
set the top left and bottom right corners of this node item
- virtual void [setBoundingBoxVisible](#) (bool visible=true, bool [controlPoints](#)=true)
show or hide the bounding box of this figure
- void [showBoundingBox](#) (bool [controlPoints](#)=true)
show the bounding box of this figure. same as setBoundingBoxVisible(true)
- void [hideBoundingBox](#) (bool [controlPoints](#)=true)
hide the bounding box of this figure. same as setBoundingBoxVisible(false)
- virtual int [type](#) () const
for enabling dynamic_cast
- virtual QList< [ConnectionGraphicsItem](#) * > [connections](#) () const
get all the connection items linked to this node
- virtual QList< [NodeGraphicsItem](#) * > [connectedNodes](#) () const
get all the nodes connected to all the connections

- virtual QList< [ConnectionGraphicsItem](#) * > [connectionsWithArrows](#) () const
get all the connection items that have an arrow associated with this node
- virtual QList< [ConnectionGraphicsItem](#) * > [connectionsWithoutArrows](#) () const
get all the connection items that do NOT have an arrow associated with this node
- virtual QList< [ConnectionGraphicsItem](#) * > [connectionsDisconnected](#) () const
get all the connection items where this node is disconnected from the main connection, e.g. modifiers
- virtual QList< [QGraphicsItem](#) * > [connectionsAsGraphicsItems](#) () const
get all the connection items linked to this node as a list of qgraphicsitems
- virtual QList< [NodeGraphicsItem](#) * > [nodesAdjacent](#) () const
get all the node items that are bordering this node
- virtual QList< [NodeGraphicsItem](#) * > [nodesUpstream](#) () const
get all the node items that are connected to this node directly or indirectly. only nodes that are coming in are selected (with arrows) Note: if the node contains more than one connections with arrows, this list returns one downstream path from the possible paths
- virtual QList< [NodeGraphicsItem](#) * > [nodesDownstream](#) () const
get all the node items that are connected to this node directly or indirectly. only nodes that are going out are selected (without arrows) Note: if the node contains more than one connections without arrows, this list returns one downstream path from the possible paths
- virtual QList< [NodeGraphicsItem](#) * > [nodesToLeft](#) () const
nodes to the left of this node in sequence
- virtual QList< [NodeGraphicsItem](#) * > [nodesToRight](#) () const
nodes to the right of this node in sequence
- virtual QList< [NodeGraphicsItem](#) * > [nodesAbove](#) () const
nodes above of this node in sequence
- virtual QList< [NodeGraphicsItem](#) * > [nodesBelow](#) () const
nodes below of this node in sequence
- virtual [Shape](#) * [tallestShape](#) () const
get the shape with greatest height inside this group graphics item
- virtual [Shape](#) * [longestShape](#) () const
get the shape with greatest width inside this group graphics item
- virtual [Shape](#) * [leftMostShape](#) () const
get the shape with lowest x value inside this group graphics item
- virtual [Shape](#) * [rightMostShape](#) () const
get the shape with largest x value inside this group graphics item

- virtual [Shape](#) * [topMostShape](#) () const
get the shape with lowest y value inside this group graphics item
- virtual [Shape](#) * [bottomMostShape](#) () const
get the shape with largest y value inside this group graphics item

Static Public Member Functions

- static [NodeGraphicsItem](#) * [cast](#) (QGraphicsItem *)
cast a graphics item to a node graphics item using qgraphicsitem_cast
- static QList< [NodeGraphicsItem](#) * > [cast](#) (const QList< QGraphicsItem * > &)
cast a list of graphics item to a list of node graphics items using qgraphicsitem_cast
- static [NodeGraphicsItem](#) * [topLevelNodeItem](#) (QGraphicsItem *item, bool ignoreControlPoints=false)
Gets the node item from one of its child items.

Public Attributes

- QString [className](#)
for safe static casting
- QString [name](#)
file where the graphics item is stored
- QSizeF [defaultSize](#)
default size for this item
- QVector< [Shape](#) * > [shapes](#)
shapes that comprise this figure
- QVector< [ControlPoint](#) * > [controlPoints](#)
control points that control the shapes in this figure
- QVector< [ControlPoint](#) * > [boundaryControlPoints](#)
set of control points that control the bounding box of this figure
- QString [groupID](#)
for identifying which scene this item belongs in

Static Public Attributes

- static const QString [CLASSNAME](#) = QString("NodeGraphicsItem")
for safe static casting

- static const int `numShapeTypes` = 4
number of different type of shapes available

Protected Member Functions

- virtual void `recomputeBoundingRect` ()
reconstruct bounding rect
- virtual qreal `getPenWidthForBoundingRect` ()
get pen width based on bounding rect

Protected Attributes

- QRectF `boundingRectangle`
bounding rectangle for the whole group
- `ItemHandle` * `itemHandle`
Tinkercell object that this drawable belongs in.
- QGraphicsRectItem * `boundingBoxItem`
the bounding box of this figure

6.44.1 Detailed Description

A simple figure made from one or more polygons. The class can be represented in an XML file.

6.44.2 Constructor & Destructor Documentation

6.44.2.1 Tinkercell::NodeGraphicsItem::NodeGraphicsItem (QGraphicsItem * *parent* = 0)

Constructor: does nothing

6.44.2.2 Tinkercell::NodeGraphicsItem::NodeGraphicsItem (const QString & *filename*, QGraphicsItem * *parent* = 0)

Construct from file using NodeGraphicsReader

6.44.2.3 Tinkercell::NodeGraphicsItem::NodeGraphicsItem (const NodeGraphicsItem & *copy*)

Copy Constructor

Copy Constructor: deep copy of all pointers

copy handle

Copy control points and shapes

6.44.2.4 TinkerCell::NodeGraphicsItem::~~NodeGraphicsItem () [virtual]

Destructor: deletes all shapes and control points.

Destructor: deletes all shapes and control points

6.44.3 Member Function Documentation

6.44.3.1 NodeGraphicsItem * TinkerCell::NodeGraphicsItem::cast (QGraphicsItem * q) [static]

cast a graphics item to a node graphics item using qgraphicsitem_cast

Parameters

*QGraphicsItem** graphics item

Returns

NodeGraphicsItem* can be 0 if the cast is invalid

Reimplemented in [TinkerCell::ArrowHeadItem](#).

6.44.3.2 QList< NodeGraphicsItem * > TinkerCell::NodeGraphicsItem::cast (const QList< QGraphicsItem * > & list) [static]

cast a list of graphics item to a list of node graphics items using qgraphicsitem_cast

Parameters

QList<QGraphicsItem>* graphics items

Returns

QList<NodeGraphicsItem*> can be empty if no cast is invalid

6.44.3.3 void TinkerCell::NodeGraphicsItem::clear () [virtual]

Clear all shapes and control points.

Parameters

void

Returns

void

6.44.3.4 NodeGraphicsItem * TinkerCell::NodeGraphicsItem::clone () const [virtual]

make a copy of this node item

make a copy of this item

Reimplemented in [TinkerCell::ArrowHeadItem](#).

6.44.3.5 `QList< NodeGraphicsItem * > Tinkercell::NodeGraphicsItem::connectedNodes () const [virtual]`

get all the nodes connected to all the connections

get all the connected nodes

6.44.3.6 `QList< QGraphicsItem * > Tinkercell::NodeGraphicsItem::connectionsAsGraphicsItems () const [virtual]`

get all the connection items linked to this node as a list of qgraphicsitems

get all the connection items linked to this node

6.44.3.7 `QList< ConnectionGraphicsItem * > Tinkercell::NodeGraphicsItem::connectionsDisconnected () const [virtual]`

get all the connection items where this node is disconnected from the main connection, e.g. modifiers

get all the connection items linked to this node

6.44.3.8 `QList< ConnectionGraphicsItem * > Tinkercell::NodeGraphicsItem::connectionsWithArrows () const [virtual]`

get all the connection items that have an arrow associated with this node

get all the connection items linked to this node

6.44.3.9 `QList< ConnectionGraphicsItem * > Tinkercell::NodeGraphicsItem::connectionsWithoutArrows () const [virtual]`

get all the connection items that do NOT have an arrow associated with this node

get all the connection items linked to this node

6.44.3.10 `void Tinkercell::NodeGraphicsItem::normalize () [virtual]`

normalizes a node graphics item so that its center is 0,0 and width*height is 10

Parameters

node item pointer to normalize

Returns

void

Parameters

NodeImage pointer to normalize

Returns

void

6.44.3.11 NodeGraphicsItem & TinkerCell::NodeGraphicsItem::operator= (const NodeGraphicsItem & copy) [virtual]

basically does the same as copy constructor

operator =: deep copy of all pointers

Copy control points and shapes

6.44.3.12 QPolygonF TinkerCell::NodeGraphicsItem::polygon () const [virtual]

gets a polygon that represents this graphicsItem

gets a polygon that is constructed by uniting all the shapes

6.44.3.13 void TinkerCell::NodeGraphicsItem::refresh () [virtual]

Updates the graphicsItem by re-initializing the vector of shapes Precondition: shapes.size > 1 Postcondition: NA.

Parameters

void

Returns

void

6.44.3.14 void TinkerCell::NodeGraphicsItem::resetBrush () [virtual]

change fill color of all shapes to the default brush

change fill color of all shapes to default

6.44.3.15 void TinkerCell::NodeGraphicsItem::resetPen () [virtual]

change outline color of all shapes to default pen

change outline color of all shapes to default

6.44.3.16 void TinkerCell::NodeGraphicsItem::resetToDefaults () [virtual]

change color, transformation, and size to defaults

change color and size to defaults

6.44.3.17 void TinkerCell::NodeGraphicsItem::setAlpha (int value) [virtual]

change alpha value for brush and pen of all shapes

change alpha value for brush of all shapes

6.44.3.18 QPainterPath TinkerCell::NodeGraphicsItem::shape () const [virtual]

gets a path that represents this graphicsItem

gets a path that is constructed by uniting all the shape paths

6.44.3.19 NodeGraphicsItem * TinkerCell::NodeGraphicsItem::topLevelNodeItem (QGraphicsItem * item, bool ignoreControlPoints = false) [static]

Gets the node item from one of its child items.

gets the node graphics item from its child item

Parameters

*QGraphicsItem** the target item

bool using true here will return the node item for a control point, otherwise control points are ignored

The documentation for this class was generated from the following files:

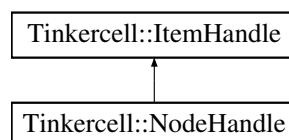
- NodeGraphicsItem.h
- NodeGraphicsItem.cpp

6.45 TinkerCell::NodeHandle Class Reference

The handles are used to bring together data and graphics items. Node Handle contains pointers to all the graphics items that belong to it, the tools that apply to this item, the data for this item, and the family that it belongs with.

```
#include <ItemHandle.h>
```

Inheritance diagram for TinkerCell::NodeHandle:



Public Member Functions

- virtual QList< [ConnectionHandle](#) * > [connections](#) () const
function that returns all the connections from all the nodes in this handle
- [NodeHandle](#) (const QString &name=QString(), [NodeFamily](#) *nodeFamily=0)

default constructor -- initialize everything

- **NodeHandle** (const **NodeHandle** ©)
copy constructor -- copies all the data (deep). graphic items are shallow copies
- virtual **NodeHandle** & **operator=** (const **NodeHandle** &)
operator =
- **NodeHandle** (**NodeFamily** *nodeFamily, **NodeGraphicsItem** *item)
constructor using initial family and graphics item
- **NodeHandle** (**NodeFamily** *nodeFamily, const QString &name=QString())
constructor using initial family and name
- virtual **ItemHandle** * **clone** () const
return a clone of this handle
- virtual **ItemFamily** * **family** () const
get the node family for this handle
- virtual void **setFamily** (**ItemFamily** *, bool useCommand=true)
set the node family for this handle

Static Public Member Functions

- static **NodeHandle** * **cast** (**ItemHandle** *)
checks if the item handle is a node handle and casts it as a node item. Returns 0 if it is not a node item
- static QList< **NodeHandle** * > **cast** (const QList< **ItemHandle** * > &)
checks if the item handles are node handles and casts then as node items. Returns QList<NodeHandle>*

Public Attributes

- **NodeFamily** * nodeFamily
node family for this node handle

Static Public Attributes

- static const int **TYPE** = 1
this number is used to identify when a handle is a node handle

6.45.1 Detailed Description

The handles are used to bring together data and graphics items. Node Handle contains pointers to all the graphics items that belong to it, the tools that apply to this item, the data for this item, and the family that it belongs with.

6.45.2 Constructor & Destructor Documentation

6.45.2.1 Tinkercell::NodeHandle::NodeHandle (NodeFamily * *nodeFamily*, NodeGraphicsItem * *item*)

constructor using initial family and graphics item

Parameters

*nodeFamily** node family

*NodeGraphicsItem** graphics item

6.45.2.2 Tinkercell::NodeHandle::NodeHandle (NodeFamily * *nodeFamily*, const QString & *name* = *QString()*)

constructor using initial family and name

Parameters

*nodeFamily** node family

QString name

6.45.3 Member Function Documentation

6.45.3.1 NodeHandle * Tinkercell::NodeHandle::cast (ItemHandle * *item*) [static]

checks if the item handle is a node handle and casts it as a node item. Returns 0 if it is not a node item

Parameters

*ItemHandle** item

6.45.3.2 QList< NodeHandle * > Tinkercell::NodeHandle::cast (const QList< ItemHandle * > & *items*) [static]

checks if the item handles are node handles and casts then as node items. Returns QList<NodeHandle*>

Parameters

Returns QList<ItemHandle*> items

6.45.3.3 ItemHandle * Tinkercell::NodeHandle::clone () const [virtual]

return a clone of this handle

Returns

ItemFamily* node handle as item handle

Reimplemented from [Tinkercell::ItemHandle](#).

6.45.3.4 `QList< ConnectionHandle * > Tinkercell::NodeHandle::connections () const` `[virtual]`

function that returns all the connections from all the nodes in this handle

Returns

`QList<ConnectionHandle*>` list of connection handles

6.45.3.5 `ItemFamily * Tinkercell::NodeHandle::family () const` `[virtual]`

get the node family for this handle

Returns

`ItemFamily*` node family as item family

Reimplemented from [Tinkercell::ItemHandle](#).

6.45.3.6 `void Tinkercell::NodeHandle::setFamily (ItemFamily * p, bool useCommand = true)` `[virtual]`

set the node family for this handle

Parameters

*NodeFamily** node family

Reimplemented from [Tinkercell::ItemHandle](#).

The documentation for this class was generated from the following files:

- `ItemHandle.h`
- `ItemHandle.cpp`

6.46 Tinkercell::PopupListWidgetDelegate Class Reference

delegate used inside the [SimpleInputWindow](#)

```
#include <AbstractInputWindow.h>
```

Public Member Functions

- `PopupListWidgetDelegate` (`QObject *parent=0`)
- `QWidget * createEditor` (`QWidget *parent, const QStyleOptionViewItem &option, const QModelIndex &index`) `const`
create the editor for the table widget delegate
- `void setEditorData` (`QWidget *editor, const QModelIndex &index`) `const`
set the data the editor for the table widget delegate

- void [setModelData](#) (QWidget *editor, QAbstractItemModel *model, const QModelIndex &index) const
set the data the editor for the table widget delegate
- void [updateEditorGeometry](#) (QWidget *editor, const QStyleOptionViewItem &option, const QModelIndex &index) const
set geometry
- bool [editorEvent](#) (QEvent *event, QAbstractItemModel *model, const QStyleOptionViewItem &option, const QModelIndex &index)
editor event

Static Public Member Functions

- static QString [displayListWidget](#) (const QStringList &list, const QString ¤t=QString(), bool *dialogOpen=0)
ask user to get a string from list of strings

Public Attributes

- [DataTable](#)< QStringList > [options](#)
options for the combo boxes. Uses line edits if empty. Uses check boxes if just one item
- bool [dialogOpen](#)
dialog is open

6.46.1 Detailed Description

delegate used inside the [SimpleInputWindow](#)

The documentation for this class was generated from the following files:

- AbstractInputWindow.h
- AbstractInputWindow.cpp

6.47 Tinkercell::PopupListWidgetDelegateDialog Class Reference

dialog for list widget

```
#include <AbstractInputWindow.h>
```

Public Slots

- void [acceptListWidget](#) (QListWidgetItem *)

6.47.1 Detailed Description

dialog for list widget

The documentation for this class was generated from the following file:

- AbstractInputWindow.h

6.48 Tinkercell::ProcessThread Class Reference

This class is used to run a process (command + args) as a separate thread as a separate thread.

```
#include <CThread.h>
```

Public Member Functions

- [ProcessThread](#) (const QString &, const QString &, [MainWindow](#) *main)
constructor -- used to initialize the main window, the command name and the args for the command
- virtual QString [output](#) () const
get the results (output stream) from the process
- virtual QString [errors](#) () const
get the errors (error stream) from the process
- virtual [~ProcessThread](#) ()
destructor -- free the library that this thread loaded

Static Public Member Functions

- static QWidget * [dialog](#) ([MainWindow](#) *, [ProcessThread](#) *, const QString &text=QString("Process"), QIcon icon=QIcon())
creates a dialog that shows the name of the running thread and a button for terminating the thread

Protected Slots

- virtual void [stopProcess](#) ()
unload the library (if loaded) and delete it

Protected Member Functions

- virtual void [run](#) ()
initializes the function pointers through the main window and then runs the target function

Protected Attributes

- QString `exe`
the name of the executable
- QString `args`
the arguments
- QString `outputStream`
the output from the process
- QString `errStream`
the error from the process
- MainWindow * `mainWindow`
Tinkercell's main window.
- QProcess `process`
Tinkercell's main window.

6.48.1 Detailed Description

This class is used to run a process (command + args) as a separate thread as a separate thread.

6.48.2 Constructor & Destructor Documentation

6.48.2.1 Tinkercell::ProcessThread::ProcessThread (const QString & *exe*, const QString & *args*, MainWindow * *main*)

constructor -- used to initialize the main window, the command name and the args for the command

Parameters

QString command
 QString arguments
 MainWindow main window

6.48.3 Member Function Documentation

6.48.3.1 QWidget * Tinkercell::ProcessThread::dialog (MainWindow * *mainWindow*, ProcessThread * *newThread*, const QString & *text* = QString("Process"), QIcon *icon* = QIcon()) [static]

creates a dialog that shows the name of the running thread and a button for terminating the thread

Parameters

MainWindow main window
 ProcessThread

QString text to display

QIcon icon to display

6.48.3.2 QString Tinkercell::ProcessThread::errors () const [virtual]

get the errors (error stream) from the process

Returns

QString output

6.48.3.3 QString Tinkercell::ProcessThread::output () const [virtual]

get the results (output stream) from the process

Returns

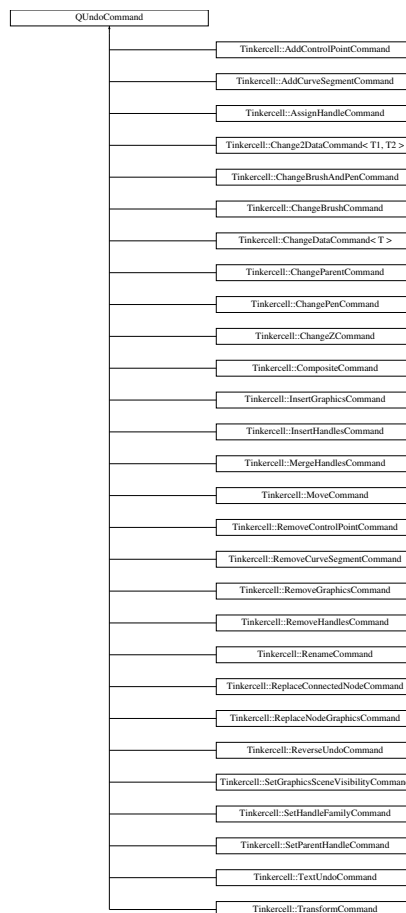
QString output

The documentation for this class was generated from the following files:

- CThread.h
- CThread.cpp

6.49 QUndoCommand Class Reference

Inheritance diagram for QUndoCommand:



The documentation for this class was generated from the following file:

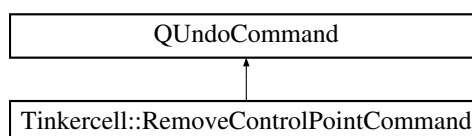
- DataTable.h

6.50 TinkerCell::RemoveControlPointCommand Class Reference

A command that removed control points. Allows undo and redo.

```
#include <UndoCommands.h>
```

Inheritance diagram for TinkerCell::RemoveControlPointCommand:



Public Member Functions

- [RemoveControlPointCommand](#) (const QString &name, [GraphicsScene](#) *scene, [ConnectionGraphicsItem::ControlPoint](#) *item)
constructor that makes the command. If added to history stack, also does redo
- [RemoveControlPointCommand](#) (const QString &name, [GraphicsScene](#) *scene, QList< [ConnectionGraphicsItem::ControlPoint](#) * > items)
constructor that makes the command. If added to history stack, also does redo
- void [redo](#) ()
Remove new control points. Control points were set in the constructor.
- void [undo](#) ()
Add new control points. Control points were set in the constructor.

Public Attributes

- QList< [ConnectionGraphicsItem::ControlPoint](#) * > [graphicsItems](#)
control points that were added
- [GraphicsScene](#) * [graphicsScene](#)
graphics scene to which control points were added
- QList< int > [listK1](#)
the position(s) at which the control points were added
- QList< int > [listK2](#)

6.50.1 Detailed Description

A command that removed control points. Allows undo and redo.

6.50.2 Constructor & Destructor Documentation

6.50.2.1 TinkerCell::RemoveControlPointCommand::RemoveControlPointCommand (const QString & name, GraphicsScene * scene, ConnectionGraphicsItem::ControlPoint * item)

constructor that makes the command. If added to history stack, also does redo

Parameters

name
graphics scene
control point(s) that have been added

Returns

void

6.50.2.2 Tinkercell::RemoveControlPointCommand::RemoveControlPointCommand (const QString & *name*, GraphicsScene * *scene*, QList<ConnectionGraphicsItem::ControlPoint * > *items*)

constructor that makes the command. If added to history stack, also does redo

Parameters

name

graphics scene

control point(s) that have been added

Returns

void

6.50.3 Member Function Documentation

6.50.3.1 void Tinkercell::RemoveControlPointCommand::redo ()

Remove new control points. Control points were set in the constructor.

Parameters

void

Returns

void

6.50.3.2 void Tinkercell::RemoveControlPointCommand::undo ()

Add new control points. Control points were set in the constructor.

Parameters

void

Returns

void

The documentation for this class was generated from the following files:

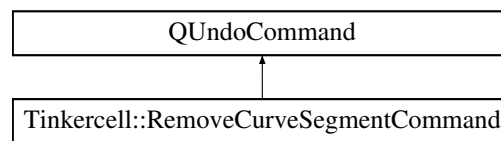
- UndoCommands.h
- UndoCommands.cpp

6.51 Tinkercell::RemoveCurveSegmentCommand Class Reference

A command that removed control points. Allows undo and redo.

```
#include <UndoCommands.h>
```

Inheritance diagram for TinkerCell::RemoveCurveSegmentCommand:



Public Member Functions

- [RemoveCurveSegmentCommand](#) (const QString &name, [GraphicsScene](#) *scene, [ConnectionGraphicsItem::ControlPoint](#) *item)
constructor that makes the command. If added to history stack, also does redo
- [RemoveCurveSegmentCommand](#) (const QString &name, [GraphicsScene](#) *scene, [ConnectionGraphicsItem](#) *connection, QList< [ConnectionGraphicsItem::ControlPoint](#) * > items)
constructor that makes the command. If added to history stack, also does redo
- void [redo](#) ()
Remove new control points. Control points were set in the constructor.
- void [undo](#) ()
Add new control points. Control points were set in the constructor.

Public Attributes

- QList< [ConnectionGraphicsItem::CurveSegment](#) > [curveSegments](#)
vector of control points that were added
- [GraphicsScene](#) * [graphicsScene](#)
graphics scene from which control points were removed
- [ConnectionGraphicsItem](#) * [connectionItem](#)
graphics item from which control points were removed
- QList< QGraphicsItem * > [parentsAtStart](#)
the nodes belonging with the control point vectors
- QList< QGraphicsItem * > [parentsAtEnd](#)

6.51.1 Detailed Description

A command that removed control points. Allows undo and redo.

6.51.2 Constructor & Destructor Documentation

6.51.2.1 Tinkercell::RemoveCurveSegmentCommand::RemoveCurveSegmentCommand (const QString & *name*, GraphicsScene * *scene*, ConnectionGraphicsItem::ControlPoint * *item*)

constructor that makes the command. If added to history stack, also does redo

Parameters

name

graphics scene

control point(s) that have been added

Returns

void

6.51.2.2 Tinkercell::RemoveCurveSegmentCommand::RemoveCurveSegmentCommand (const QString & *name*, GraphicsScene * *scene*, ConnectionGraphicsItem * *connection*, QList< ConnectionGraphicsItem::ControlPoint * > *items*)

constructor that makes the command. If added to history stack, also does redo

Parameters

name

graphics scene

control point(s) that have been added

Returns

void

6.51.3 Member Function Documentation

6.51.3.1 void Tinkercell::RemoveCurveSegmentCommand::redo ()

Remove new control points. Control points were set in the constructor.

Parameters

void

Returns

void

6.51.3.2 void TinkerCell::RemoveCurveSegmentCommand::undo ()

Add new control points. Control points were set in the constructor.

Parameters

void

Returns

void

The documentation for this class was generated from the following files:

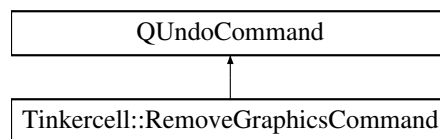
- UndoCommands.h
- UndoCommands.cpp

6.52 TinkerCell::RemoveGraphicsCommand Class Reference

this command performs an removal and allows redo/undo of that removal

```
#include <UndoCommands.h>
```

Inheritance diagram for TinkerCell::RemoveGraphicsCommand:



Public Member Functions

- [RemoveGraphicsCommand](#) (const QString &name, QGraphicsItem *item, bool updateDataFields=true)
constructor
- [RemoveGraphicsCommand](#) (const QString &name, const QList< QGraphicsItem * > &items, bool updateDataFields=true)
constructor
- void [redo](#) ()
redo the change
- void [undo](#) ()
undo the change

6.52.1 Detailed Description

this command performs an removal and allows redo/undo of that removal

6.52.2 Constructor & Destructor Documentation

6.52.2.1 Tinkercell::RemoveGraphicsCommand::RemoveGraphicsCommand (const QString & name, QGraphicsItem * item, bool updateDataFields = true)

constructor

Parameters

- QString* name of command
- GraphicsScene** where change happened
- QGraphicsItem** item that is removed
- bool* update data of other items where removed items might occur (default=true)

6.52.2.2 Tinkercell::RemoveGraphicsCommand::RemoveGraphicsCommand (const QString & name, const QList< QGraphicsItem * > & items, bool updateDataFields = true)

constructor

Parameters

- QString* name of command
- GraphicsScene** where change happened
- QList<QGraphicsItem*>&* items that are removed
- bool* update data of other items where removed items might occur (default=true)

The documentation for this class was generated from the following files:

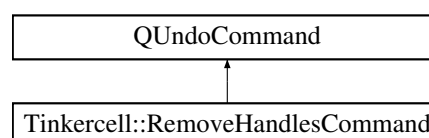
- UndoCommands.h
- UndoCommands.cpp

6.53 Tinkercell::RemoveHandlesCommand Class Reference

this command inserts new handles to a [NetworkHandle](#)

```
#include <UndoCommands.h>
```

Inheritance diagram for Tinkercell::RemoveHandlesCommand:



Public Member Functions

- [RemoveHandlesCommand](#) ([TextEditor](#) *, const QList< [ItemHandle](#) * > &, bool updateDataFields=true)

constructor

- [RemoveHandlesCommand](#) ([TextEditor](#) *, [ItemHandle](#) *, bool updateDataFields=true)

constructor

- void [redo](#) ()

redo the change

- void [undo](#) ()

undo the change

6.53.1 Detailed Description

this command inserts new handles to a [NetworkHandle](#)

6.53.2 Constructor & Destructor Documentation

6.53.2.1 Tinkercell::RemoveHandlesCommand::RemoveHandlesCommand ([TextEditor](#) * *editor*, const [QList](#)< [ItemHandle](#) * > & *list*, bool *updateDataFields* = *true*)

constructor

Parameters

*TextEditor** window where items are deleted

QList<*ItemHandle**> deleted items

bool update data of other items where removed items might occur (default=true)

6.53.2.2 Tinkercell::RemoveHandlesCommand::RemoveHandlesCommand ([TextEditor](#) * *editor*, [ItemHandle](#) * *h*, bool *updateDataFields* = *true*)

constructor

Parameters

*TextEditor** window where items are deleted

*ItemHandle** deleted item

bool update data of other items where removed items might occur (default=true)

The documentation for this class was generated from the following files:

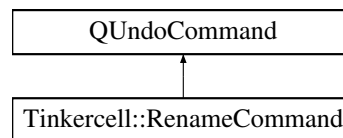
- [UndoCommands.h](#)
- [UndoCommands.cpp](#)

6.54 Tinkercell::RenameCommand Class Reference

this command changes the name of the handle of an item. important: use full name of the items!

```
#include <UndoCommands.h>
```

Inheritance diagram for Tinkercell::RenameCommand:



Public Member Functions

- **RenameCommand** (const QString &name, [NetworkHandle](#) *, const QList< [ItemHandle](#) * > &allItems, const QString &oldname, const QString &newname, bool forceUnique=true)
constructor
- **RenameCommand** (const QString &name, [NetworkHandle](#) *, const QString &oldname, const QString &newname, bool forceUnique=true)
constructor
- **RenameCommand** (const QString &name, [NetworkHandle](#) *, const QList< [ItemHandle](#) * > &allItems, const QList< QString > &oldname, const QList< QString > &newname, bool forceUnique=true)
constructor
- **RenameCommand** (const QString &name, [NetworkHandle](#) *, const QList< QString > &oldname, const QList< QString > &newname, bool forceUnique=true)
constructor
- **RenameCommand** (const QString &name, [NetworkHandle](#) *, [ItemHandle](#) *itemHandle, const QString &newname, bool forceUnique=true)
constructor
- **RenameCommand** (const QString &name, [NetworkHandle](#) *, const QList< [ItemHandle](#) * > &allItems, [ItemHandle](#) *item, const QString &newname, bool forceUnique=true)
constructor
- **RenameCommand** (const QString &name, [NetworkHandle](#) *, const QList< [ItemHandle](#) * > &itemhandles, const QList< QString > &newnames, bool forceUnique=true)
constructor
- **RenameCommand** (const QString &name, [NetworkHandle](#) *, const QList< [ItemHandle](#) * > &allItems, const QList< [ItemHandle](#) * > &itemhandles, const QList< QString > &newnames, bool forceUnique=true)
constructor
- void **redo** ()
- void **undo** ()

Static Public Member Functions

- static void **findReplaceAllHandleData** (QList< [ItemHandle](#) * > &allItems, const QString &oldName, const QString &newName)
- static void **substituteString** (QString &targetValue, const QString &oldName, const QString &newName)

6.54.1 Detailed Description

this command changes the name of the handle of an item. important: use full name of the items!

6.54.2 Constructor & Destructor Documentation

6.54.2.1 Tinkercell::RenameCommand::RenameCommand (const QString & *name*, NetworkHandle * *net*, const QList< ItemHandle * > & *allItems*, const QString & *oldname*, const QString & *newname*, bool *forceUnique* = *true*)

constructor

Parameters

QString name of command

[NetworkHandle](#) * network

QList affected items

QString old name

QString new name

bool make sure that the new names are unique (default = true). Use false if you already made this check or want to rename to something that already exists

6.54.2.2 Tinkercell::RenameCommand::RenameCommand (const QString & *name*, NetworkHandle * *net*, const QString & *oldname*, const QString & *newname*, bool *forceUnique* = *true*)

constructor

Parameters

QString name of command

[NetworkHandle](#) * network

QString old name

QString new name

bool make sure that the new names are unique (default = true). Use false if you already made this check or want to rename to something that already exists

6.54.2.3 Tinkercell::RenameCommand::RenameCommand (const QString & name, NetworkHandle * net, const QList< ItemHandle * > & allItems, const QList< QString > & oldname, const QList< QString > & newname, bool forceUnique = true)

constructor

Parameters

QString name of command

NetworkHandle * network

QList affected items

QString old name

QString new name

bool make sure that the new names are unique (default = true). Use false if you already made this check or want to rename to something that already exists

6.54.2.4 Tinkercell::RenameCommand::RenameCommand (const QString & name, NetworkHandle * net, const QList< QString > & oldname, const QList< QString > & newname, bool forceUnique = true)

constructor

Parameters

QString name of command

NetworkHandle * network

QString old name

QString new name

bool make sure that the new names are unique (default = true). Use false if you already made this check or want to rename to something that already exists

6.54.2.5 Tinkercell::RenameCommand::RenameCommand (const QString & name, NetworkHandle * net, ItemHandle * itemHandle, const QString & newname, bool forceUnique = true)

constructor

Parameters

QString name of command

NetworkHandle * network

*ItemHandle** target item handle

QString new name

bool make sure that the new names are unique (default = true). Use false if you already made this check or want to rename to something that already exists

6.54.2.6 TinkerCell::RenameCommand::RenameCommand (const QString & name, NetworkHandle * net, const QList< ItemHandle * > & allItems, ItemHandle * item, const QString & newname, bool forceUnique = true)

constructor

Parameters

QString name of command

NetworkHandle * network

QList<ItemHandle>&* all the items to modify if they contain the new name

*ItemHandle** target item

QString new name

bool make sure that the new names are unique (default = true). Use false if you already made this check or want to rename to something that already exists

6.54.2.7 TinkerCell::RenameCommand::RenameCommand (const QString & name, NetworkHandle * net, const QList< ItemHandle * > & itemhandles, const QList< QString > & newnames, bool forceUnique = true)

constructor

Parameters

QString name of command

NetworkHandle * network

QList<ItemHandle>&* target items

QList<QString> new names (one for each item)

bool make sure that the new names are unique (default = true). Use false if you already made this check or want to rename to something that already exists

6.54.2.8 TinkerCell::RenameCommand::RenameCommand (const QString & name, NetworkHandle * net, const QList< ItemHandle * > & allItems, const QList< ItemHandle * > & itemhandles, const QList< QString > & newnames, bool forceUnique = true)

constructor

Parameters

QString name of command

NetworkHandle * network

QList<ItemHandle>&* all the items to modify if they contain the new name

QList<ItemHandle>&* target items

QList<QString> new names (one for each item)

bool make sure that the new names are unique (default = true). Use false if you already made this check or want to rename to something that already exists

The documentation for this class was generated from the following files:

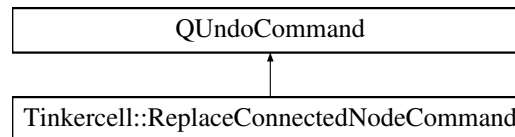
- UndoCommands.h
- UndoCommands.cpp

6.55 Tinkercell::ReplaceConnectedNodeCommand Class Reference

this command replaces one node item in a connection item with another

```
#include <UndoCommands.h>
```

Inheritance diagram for Tinkercell::ReplaceConnectedNodeCommand:



Public Member Functions

- [ReplaceConnectedNodeCommand](#) (const QString &name, [ConnectionGraphicsItem](#) *, [NodeGraphicsItem](#) *oldNode, [NodeGraphicsItem](#) *newNode)

constructor

- void **redo** ()
- void **undo** ()

6.55.1 Detailed Description

this command replaces one node item in a connection item with another

6.55.2 Constructor & Destructor Documentation

- #### 6.55.2.1 Tinkercell::ReplaceConnectedNodeCommand::ReplaceConnectedNodeCommand (const QString & *name*, [ConnectionGraphicsItem](#) * *c*, [NodeGraphicsItem](#) * *oldNode*, [NodeGraphicsItem](#) * *newNode*)

constructor

Parameters

QString name of command

*ConnectionGraphicsItem** connection where the nodes will be swapped

*NodeGraphicsItem** node to replace (old node)

*NodeGraphicsItem** new node

The documentation for this class was generated from the following files:

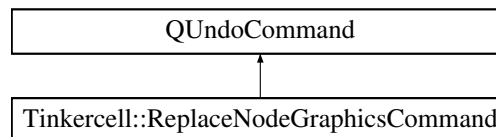
- UndoCommands.h
- UndoCommands.cpp

6.56 TinkerCell::ReplaceNodeGraphicsCommand Class Reference

this command can be used to replace the graphical representation of a node from an xml file

```
#include <UndoCommands.h>
```

Inheritance diagram for TinkerCell::ReplaceNodeGraphicsCommand:



Public Member Functions

- [ReplaceNodeGraphicsCommand](#) (const QString &, [NodeGraphicsItem](#) *, const QString &, bool transform=true)

constructor

- [ReplaceNodeGraphicsCommand](#) (const QString &, const QList< [NodeGraphicsItem](#) * > &, const QList< QString > &, bool transform=true)

constructor

- void **undo** ()
- void **redo** ()

6.56.1 Detailed Description

this command can be used to replace the graphical representation of a node from an xml file

6.56.2 Constructor & Destructor Documentation

6.56.2.1 TinkerCell::ReplaceNodeGraphicsCommand::ReplaceNodeGraphicsCommand (const QString & *text*, [NodeGraphicsItem](#) * *node*, const QString & *filename*, bool *transform* = *true*)

constructor

Parameters

QString name of command

*NodeGraphicsItem** the target node

QString xml file name

bool whether or not to transform the new graphics item to the original item's angle and size

6.56.2.2 TinkerCell::ReplaceNodeGraphicsCommand::ReplaceNodeGraphicsCommand (const QString & text, const QList< NodeGraphicsItem * > & nodes, const QList< QString > & filenames, bool transform = true)

constructor

Parameters

QString name of command

QList<NodeGraphicsItem>* the target nodes

QStringList xml file names

bool whether or not to transform the new graphics item to the original item's angle and size

The documentation for this class was generated from the following files:

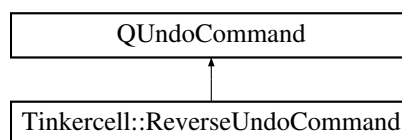
- UndoCommands.h
- UndoCommands.cpp

6.57 TinkerCell::ReverseUndoCommand Class Reference

this command can be used to invert another undo command (i.e. flip the redo/undo)

```
#include <UndoCommands.h>
```

Inheritance diagram for TinkerCell::ReverseUndoCommand:



Public Member Functions

- [ReverseUndoCommand](#) (const QString &, [QUndoCommand](#) *, bool deleteCommand=true)
constructor

- void **redo** ()
- void **undo** ()

Public Attributes

- [QUndoCommand](#) * **command**
- bool **deleteCommand**

6.57.1 Detailed Description

this command can be used to invert another undo command (i.e. flip the redo/undo)

6.57.2 Constructor & Destructor Documentation

6.57.2.1 TinkerCell::ReverseUndoCommand::ReverseUndoCommand (const QString & name, QUndoCommand * cmd, bool deleteCommand = true)

constructor

Parameters

QString name of command

QList<QUndoCommand>&* the command to invert

bool whether or not to delete the inverted command (true = DO delete)

The documentation for this class was generated from the following files:

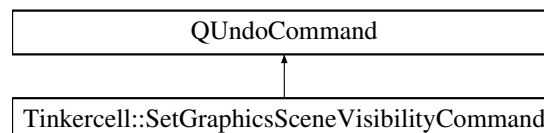
- UndoCommands.h
- UndoCommands.cpp

6.58 TinkerCell::SetGraphicsSceneVisibilityCommand Class Reference

this command is used to hide graphics items. Hidden graphics items will be part (unless their handles are also hidden) of the network but not visible on the screen.

```
#include <UndoCommands.h>
```

Inheritance diagram for TinkerCell::SetGraphicsSceneVisibilityCommand:



Public Member Functions

- [SetGraphicsSceneVisibilityCommand](#) (const QString &name, const QList< QGraphicsItem * > &, const QList< bool > &)
constructor
- [SetGraphicsSceneVisibilityCommand](#) (const QString &name, QGraphicsItem *, bool)
constructor
- [SetGraphicsSceneVisibilityCommand](#) (const QString &name, const QList< QGraphicsItem * > &, bool)
constructor
- void [redo](#) ()
redo parent change

- void [undo](#) ()
undo parent change

6.58.1 Detailed Description

this command is used to hide graphics items. Hidden graphics items will be part (unless their handles are also hidden) of the network but not visible on the screen.

The documentation for this class was generated from the following files:

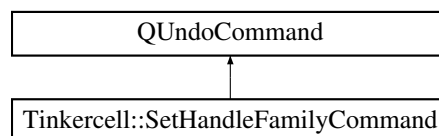
- UndoCommands.h
- UndoCommands.cpp

6.59 Tinkercell::SetHandleFamilyCommand Class Reference

this command is used to hide graphics items. Hidden graphics items will be part (unless their handles are also hidden) of the network but not visible on the screen.

```
#include <UndoCommands.h>
```

Inheritance diagram for Tinkercell::SetHandleFamilyCommand:



Public Member Functions

- [SetHandleFamilyCommand](#) (const QString &name, const QList< [ItemHandle](#) * > &, const QList< [ItemFamily](#) * > &)
constructor
- [SetHandleFamilyCommand](#) (const QString &name, [ItemHandle](#) *, [ItemFamily](#) *)
constructor
- void [redo](#) ()
redo parent change
- void [undo](#) ()
undo parent change

Friends

- class [NetworkHandle](#)

6.59.1 Detailed Description

this command is used to hide graphics items. Hidden graphics items will be part (unless their handles are also hidden) of the network but not visible on the screen.

The documentation for this class was generated from the following files:

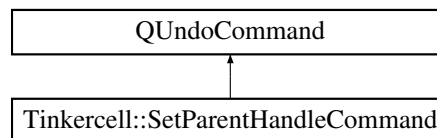
- UndoCommands.h
- UndoCommands.cpp

6.60 TinkerCell::SetParentHandleCommand Class Reference

this command assigns parent(s) to one or more handles

```
#include <UndoCommands.h>
```

Inheritance diagram for TinkerCell::SetParentHandleCommand:



Public Member Functions

- [SetParentHandleCommand](#) (const QString &name, [NetworkHandle](#) *, [ItemHandle](#) *child, [ItemHandle](#) *parent)
constructor
- [SetParentHandleCommand](#) (const QString &name, [NetworkHandle](#) *, const QList< [ItemHandle](#) * > &children, [ItemHandle](#) *parent)
constructor
- [SetParentHandleCommand](#) (const QString &name, [NetworkHandle](#) *, const QList< [ItemHandle](#) * > &children, const QList< [ItemHandle](#) * > &parents)
constructor
- [~SetParentHandleCommand](#) ()
destructor
- void [redo](#) ()
redo parent change
- void [undo](#) ()
undo parent change

Friends

- class [NetworkHandle](#)

6.60.1 Detailed Description

this command assigns parent(s) to one or more handles

The documentation for this class was generated from the following files:

- UndoCommands.h
- UndoCommands.cpp

6.61 Tinkercell::NodeGraphicsItem::Shape Class Reference

A closed polygon path made from arcs, lines, and beziers.

```
#include <NodeGraphicsItem.h>
```

Public Types

- enum { **Type** = UserType + 3 }
for enabling dynamic_cast

Public Member Functions

- [Shape](#) ([NodeGraphicsItem](#) *drawable_ptr=0, [QGraphicsItem](#) *parent=0, [QGraphicsScene](#) *scene=0)
- [Shape](#) (const [Shape](#) ©)
- virtual [Shape](#) & [operator=](#) (const [Shape](#) ©)
- void [refresh](#) ()
Generates a new polygon using the points and types vectors Precondition: points.size > 1 Postcondition: NA.
- bool [isClosed](#) () const
Checks if the polygon is closed.
- virtual [QPainterPath](#) [shape](#) () const
gets a path that represents this shape
- virtual [QRectF](#) [boundingRect](#) () const
bounding rect
- virtual int [type](#) () const
for enabling dynamic_cast

Public Attributes

- [QBrush](#) [defaultBrush](#)
permanent brush for this control point

- QPen [defaultPen](#)
permanent pen for this control point
- NodeGraphicsItem * [nodeItem](#)
paint method. Call's parent's paint after setting antialiasing to true
- bool [negative](#)
- QVector< [ControlPoint](#) * > [controlPoints](#)
control points defining this shape
- QVector< qreal > [parameters](#)
thickness, arc angles, etc.
- QVector< [ShapeType](#) > [types](#)
types of shapes to draw using the control points
- QPolygonF [polygon](#)
the polygon constructed from controls and types vectors
- QPainterPath [path](#)
the path constructed from controls and types vectors
- QPair< QPointF, QPointF > [gradientPoints](#)
start and stop coordinates for gradient fill

Protected Member Functions

- virtual void [recomputeBoundingRect](#) ()
reconstruct bounding rect

Protected Attributes

- QRectF [boundingRectangle](#)
bounding reactangle for this shape

6.61.1 Detailed Description

A closed polygon path made from arcs, lines, and beziers.

6.61.2 Constructor & Destructor Documentation

6.61.2.1 TinkerCell::NodeGraphicsItem::Shape::Shape (NodeGraphicsItem * *idrawable_ptr* = 0, QGraphicsItem * *parent* = 0, QGraphicsScene * *scene* = 0)

Constructor: sets angle to 0 and scale to 1

6.61.2.2 TinkerCell::NodeGraphicsItem::Shape::Shape (const Shape & copy)

Copy Constructor

Copy Constructor : shallow copy of all vectors

6.61.3 Member Function Documentation

6.61.3.1 QRectF TinkerCell::NodeGraphicsItem::Shape::boundingRect () const [virtual]

bounding rect

bounding rectangle

6.61.3.2 NodeGraphicsItem::Shape & TinkerCell::NodeGraphicsItem::Shape::operator= (const Shape & copy) [virtual]

Copy operator

operator = shallow copy of all vectors

6.61.3.3 void TinkerCell::NodeGraphicsItem::Shape::refresh ()

Generates a new polygon using the points and types vectors Precondition: points.size > 1 Postcondition: NA.

paint method. Call's parent's paint after setting antialiasing to true

Parameters

void

Returns

void

Generates a new polygon using the points and types vectors Precondition: controlPoints.size > 1 Postcondition: NA

Parameters

void

Returns

void

6.61.3.4 QPainterPath TinkerCell::NodeGraphicsItem::Shape::shape () const [virtual]

gets a path that represents this shape

gets a path that represents this graphicsItem

6.61.4 Member Data Documentation

6.61.4.1 bool TinkerCell::NodeGraphicsItem::Shape::negative

is this a negative (clip out) shape

6.61.4.2 NodeGraphicsItem* TinkerCell::NodeGraphicsItem::Shape::nodeItem

paint method. Call's parent's paint after setting antialiasing to true

the [NodeGraphicsItem](#) that this shape belongs in

The documentation for this class was generated from the following files:

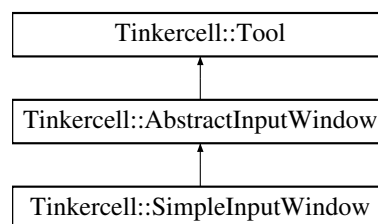
- NodeGraphicsItem.h
- NodeGraphicsItem.cpp

6.62 TinkerCell::SimpleInputWindow Class Reference

Used to create an input window that can receive user inputs for C plugins.

```
#include <AbstractInputWindow.h>
```

Inheritance diagram for TinkerCell::SimpleInputWindow:



Public Slots

- virtual void [exec](#) ()
Executes the [CThread](#).

Static Public Member Functions

- static [SimpleInputWindow](#) * [CreateWindow](#) ([MainWindow](#) *main, const QString &title, const QString &libraryFile, const QString &funcName, const [DataTable](#)< qreal > &)
Create a simple input window to run a [CThread](#). The window can be used to fill in an input matrix.
- static [SimpleInputWindow](#) * [CreateWindow](#) ([CThread](#) *cthread, const QString &title, void(*f)(tc_matrix), const [DataTable](#)< qreal > &)
creates a docking window in TinkerCell's mainwindow that can receive inputs from user and run a function in a separate thread

- static [SimpleInputWindow](#) * [CreateWindow](#) ([MainWindow](#) *main, const QString &title, const QString &funcName, const [DataTable](#)< qreal > &)

Create a simple input window to run a script function. When the play button is pressed, this window will execute a command in the command window. The command will be f(arg1,arg2...), where f is the function name and arg1,arg2... are the user provided arguments in the input window.

- static void [AddOptions](#) (const QString &title, int i, int j, const QStringList &options)
add a list of options (combo box) to an existing input window
- static void [AddOptions](#) ([SimpleInputWindow](#) *, int i, int j, const QStringList &options)
add a list of options (combo box) to an existing input window

Protected Slots

- virtual void [dataChanged](#) (int, int)
updates the input matrix when user changes the table
- virtual void [addRow](#) ()
add a row to the input matrix
- virtual void [removeRow](#) ()
remove a row from the input matrix
- virtual void [comboBoxChanged](#) (int)
updates the input matrix when user changes the combo boxes

Protected Member Functions

- [SimpleInputWindow](#) ([MainWindow](#) *main, const QString &title, const QString &dllName, const QString &funcName, const [DataTable](#)< qreal > &)
constructor that creates a docking window in Tinkercell's mainwindow that can receive inputs from user and run a function in a separate thread
- [SimpleInputWindow](#) ([CThread](#) *thread, const QString &title, void(*f)(tc_matrix), const [DataTable](#)< qreal > &)
constructor that creates a docking window in Tinkercell's mainwindow that can receive inputs from user and run a function in a separate thread
- [SimpleInputWindow](#) ([MainWindow](#) *main, const QString &title, const [DataTable](#)< qreal > &)
constructor that creates a docking window in Tinkercell's mainwindow that can receive inputs from user and run a function in a separate thread
- [SimpleInputWindow](#) ()
constructor -- does nothing
- [SimpleInputWindow](#) (const [SimpleInputWindow](#) &)
copy constructor

- virtual void [setupDisplay](#) (const [DataTable](#)< qreal > &)
reinitialize the contents on the input window
- void [leaveEvent](#) (QEvent *event)
make the window transparent when mouse exits the window
- void [enterEvent](#) (QEvent *event)
make the window transparent when mouse exits the window

Protected Attributes

- [DataTable](#)< qreal > [dataTable](#)
the input matrix
- QWidget [tableWidget](#)
the table displaying the input matrix
- QList< QComboBox * > [comboBoxes](#)
combo boxes used in input window
- [PopupListWidgetDelegate](#) [delegate](#)
the item delegate that is used to change values in the input window
- QString [scriptCommand](#)
command that will be run when the play button is pressed (might be empty if a C or C++ function is the target function)

Static Protected Attributes

- static QHash< QString, [SimpleInputWindow](#) * > [inputWindows](#)
the set of all simple input windows

6.62.1 Detailed Description

Used to create an input window that can receive user inputs for C plugins.

6.62.2 Constructor & Destructor Documentation

- 6.62.2.1** [TinkerCell::SimpleInputWindow::SimpleInputWindow](#) ([MainWindow](#) * *main*, const [QString](#) & *title*, const [QString](#) & *dllName*, const [QString](#) & *funcName*, const [DataTable](#)< qreal > & *data*) [[protected](#)]

constructor that creates a docking window in TinkerCell's mainwindow that can receive inputs from user and run a function in a separate thread

Parameters*Main Window**QString* title*QString* dynamic library file*QString* function to run inside library*QDataTable<qreal>* input table and its default values

6.62.2.2 Tinkercell::SimpleInputWindow::SimpleInputWindow (CThread * *thread*, const QString & *title*, void(*)(*tc_matrix*) *f*, const DataTable<qreal> & *data*) [protected]

constructor that creates a docking window in Tinkercell's mainwindow that can receive inputs from user and run a function in a separate thread

Parameters*CThread* * existing thread with the library containing the function*QString* title*inputtc_matrixFunction** function that is triggered by the run button in the input window*QDataTable<qreal>* input table and its default values

6.62.2.3 Tinkercell::SimpleInputWindow::SimpleInputWindow (MainWindow * *main*, const QString & *title*, const DataTable<qreal> & *data*) [protected]

constructor that creates a docking window in Tinkercell's mainwindow that can receive inputs from user and run a function in a separate thread

Parameters*QString* title*QDataTable<qreal>* input table and its default values**6.62.3 Member Function Documentation**

6.62.3.1 void Tinkercell::SimpleInputWindow::AddOptions (const QString & *title*, int *i*, int *j*, const QStringList & *options*) [static]

add a list of options (combo box) to an existing input window

Parameters*QString* title*int* row*int* column*QStringList* options

6.62.3.2 void Tinkercell::SimpleInputWindow::AddOptions (SimpleInputWindow * win, int i, int j, const QStringList & options) [static]

add a list of options (combo box) to an existing input window

Parameters

*SimpleInputWindow**

int row

int column

QStringList options

6.62.3.3 SimpleInputWindow * Tinkercell::SimpleInputWindow::CreateWindow (CThread * cthread, const QString & title, void(*) (tc_matrix) f, const DataTable< qreal > & data) [static]

creates a docking window in Tinkercell's mainwindow that can receive inputs from user and run a function in a separate thread

Parameters

CThread * existing thread with the library containing the function

QString title

*itc_matrixFunction** function that is triggered by the run button in the input window

QDataTable< qreal > input table and its default values

Returns

SimpleInputWindow* pointer to the new or existing window

6.62.3.4 SimpleInputWindow * Tinkercell::SimpleInputWindow::CreateWindow (MainWindow * main, const QString & title, const QString & libraryFile, const QString & funcName, const DataTable< qreal > & data) [static]

Create a simple input window to run a *CThread*. The window can be used to fill in an input matrix.

Parameters

MainWindow

QString title

QString dynamic library file (will first search if already loaded in *MainWindow*)

QString function to run inside library

DataTable< double > inputs

Returns

SimpleInputWindow* pointer to the new or existing window

6.62.3.5 SimpleInputWindow * Tinkercell::SimpleInputWindow::CreateWindow (MainWindow * main, const QString & title, const QString & funcName, const DataTable< qreal > & data) [static]

Create a simple input window to run a script function. When the play button is pressed, this window will execute a command in the command window. The command will be f(arg1,arg2...), where f is the function name and arg1,arg2... are the user provided arguments in the input window.

Parameters

[MainWindow](#)

QString title

QString function name

DataTable<double> inputs

Returns

SimpleInputWindow* pointer to the new or existing window

6.62.3.6 void Tinkercell::SimpleInputWindow::exec () [virtual, slot]

Executes the [CThread](#).

See also

[CThread](#)

Reimplemented from [Tinkercell::AbstractInputWindow](#).

The documentation for this class was generated from the following files:

- AbstractInputWindow.h
- AbstractInputWindow.cpp

6.63 Tinkercell::SymbolsTable Class Reference

The symbols table is updated every time the scene or text editor changes. The symbols table contains the list of item names and [ItemHandle](#) pointers as well as names and pointers to each data entry in each item.

```
#include <SymbolsTable.h>
```

Public Member Functions

- [SymbolsTable](#) ([NetworkHandle](#) *)
constructor
- virtual void [update](#) (int n=0)
update the symbols table
- virtual bool [isValidPointer](#) (void *) const

checks whether the given item handle pointer is valid

- virtual QList< [ItemHandle](#) * > [allHandlesSortedByFamily](#) () const
get list of all items sorted according to family
- virtual QList< [ItemHandle](#) * > [allHandlesSortedByName](#) () const
get list of all items sorted according to their full name

Public Attributes

- QHash< QString, [ItemHandle](#) * > [uniqueHandlesWithDot](#)
handle names and the corresponding handles. This hash stores the unique full names, such a M.A and M_A
- QHash< QString, [ItemHandle](#) * > [uniqueHandlesWithUnderscore](#)
- QHash< QString, [ItemHandle](#) * > [nonuniqueHandles](#)
handle names and the corresponding handles. This hash stores the the non-unique names, such as A. Therefore the hash may contain multiple values for the same key (see QHash documentation)
- QHash< QString, QPair< [ItemHandle](#) *, QString > > [uniqueDataWithDot](#)
row or column name and the corresponding handle and tool in which the row or column name belongs. Stores full names only. For example, if A.k0 is a data item, then this table will contain A.k0 and A_k0. All entries are unique.
- QHash< QString, QPair< [ItemHandle](#) *, QString > > [uniqueDataWithUnderscore](#)
- QHash< QString, QPair< [ItemHandle](#) *, QString > > [nonuniqueData](#)
row or column name and the corresponding handle and tool in which the row or column name belongs. Stores just the row or column name. For example, if A.k0 is a data item, then this table will contain k0. The individual, non-unique, names such as k0 may have multiple hash values for the same hash key (see QHash documentation).
- QHash< QString, [ItemHandle](#) * > [handlesByFamily](#)
this hash contains all the list of items belonging in each family. The items are listed under their family only and not under their parent families. For example, you will not find an item of family "Elephant" under the "Mammals" key. You will have to specifically search under "Elephant" and use ItemFamily's isA method to find out that it is also a "Mammal"

Protected Member Functions

- virtual void [update](#) (const QList< [ItemHandle](#) * > &)
update the symbols table

Protected Attributes

- [NetworkHandle](#) * [network](#)
the network that this symbols table belongs with
- [ItemHandle](#) [globalHandle](#)

This is a special item handle that does not represent any item on the scene. It is used to store "global" data.

- `QHash< void *, QString >` [handlesAddress](#)
addresses of all handles

Friends

- class `NetworkHandle`

6.63.1 Detailed Description

The symbols table is updated every time the scene or text editor changes. The symbols table contains the list of item names and [ItemHandle](#) pointers as well as names and pointers to each data entry in each item.

6.63.2 Constructor & Destructor Documentation

6.63.2.1 `Tinkercell::SymbolsTable::SymbolsTable (NetworkHandle * net)`

constructor

Parameters

*NetworkWindow** network that this symbol table belongs in

The documentation for this class was generated from the following files:

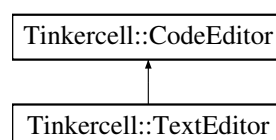
- `SymbolsTable.h`
- `SymbolsTable.cpp`

6.64 Tinkercell::TextEditor Class Reference

This is the window that allows used to construct networks using text, as opposed to graphics, which is done by [GraphicsScene](#). The [TextEditor](#) requires a supporting tool that parses the text and calls the `itemsInserted` or `itemsRemoved` methods. Without a supporting parser tool, the [TextEditor](#) will not do anything.

```
#include <TextEditor.h>
```

Inheritance diagram for `Tinkercell::TextEditor`:



Public Slots

- virtual void [popOut \(\)](#)

calls main window's popOut

- virtual void [popIn](#) ()

calls main window's popIn

- virtual void [undo](#) ()

undo last edit

- virtual void [redo](#) ()

redo last undo

- virtual void [selectAll](#) ()

select all text

- virtual void [copy](#) ()

copy selected text

- virtual void [cut](#) ()

cut selected text

- virtual void [paste](#) ()

paste text from clipboard

- virtual void [print](#) (QPrinter *printer)

print text

Signals

- void [textChanged](#) ([TextEditor](#) *, const QString &, const QString &, const QString &)

some text inside this editor has been changed

- void [lineChanged](#) ([TextEditor](#) *, int, const QString &)

the cursor has moved to a different line

- void [itemsInserted](#) ([NetworkHandle](#) *, const QList< [ItemHandle](#) * > &)

signal that is emitted when items are inserted in this [TextEditor](#).

- void [itemsRemoved](#) ([NetworkHandle](#) *, const QList< [ItemHandle](#) * > &)

signal that is emitted when items are removed from this [TextEditor](#).

- void [parse](#) ([TextEditor](#) *)

request to parse the text in the current text editor

Public Member Functions

- [TextEditor](#) ([NetworkHandle](#) *, [QWidget](#) *parent=0)
default constructor
- [~TextEditor](#) ()
destructor -- removes all the text items
- void [insert](#) ([ItemHandle](#) *)
insert a text item
- void [insert](#) (const [QList](#)< [ItemHandle](#) * > &)
insert text items
- void [remove](#) ([ItemHandle](#) *)
remove an item
- void [remove](#) (const [QList](#)< [ItemHandle](#) * > &)
remove text items
- void [setItems](#) (const [QList](#)< [ItemHandle](#) * > &)
clear existing items and insert new items
- [QList](#)< [ItemHandle](#) * > & [items](#) ()
all the items represented by the text in this [TextEditor](#)
- void [push](#) ([QUndoCommand](#) *)
push a command to the undo/redo stack
- [QString](#) [selectedText](#) () const
gets the selected text
- [MainWindow](#) * [mainWindow](#) () const
the main window containing this network
- [ConsoleWindow](#) * [console](#) () const
same as network->mainWindow->[console\(\)](#)
- [ItemHandle](#) * [localHandle](#) () const
same as networkWindow->handle
- [ItemHandle](#) * [globalHandle](#) () const
same as network->[globalHandle\(\)](#)

Public Attributes

- [QMenu](#) * [contextSelectionMenu](#)
the context menu that is shown during right-click event on a text editor with text selected. Plugins can add new actions to this menu.

- QMenu * [contextEditorMenu](#)
the context menu that is shown during right-click event on a text editor with no text selected. Plugins can add new actions to this menu.
- [NetworkHandle](#) * [network](#)
the network handle represented in this text editor
- [NetworkWindow](#) * [networkWindow](#)
the network window containing this text editor

Static Public Attributes

- static bool **SideBarEnabled** = true

Protected Member Functions

- virtual void [keyPressEvent](#) (QKeyEvent *event)
listens to keyboard events in order to determine when the current line has changed
- virtual void [mousePressEvent](#) (QMouseEvent *event)
listens to mouse events just to activate this window
- virtual void [contextMenuEvent](#) (QContextMenuEvent *event)
creates context menu with actions in the contextMenu member
- virtual void [mouseReleaseEvent](#) (QMouseEvent *event)
emits line changed and text changed if needed

Protected Attributes

- int [prevBlockNumber](#)
previously accessed line number. This is to keep track of when a line is modified
- int [changedBlockNumber](#)
current line number. This is to keep track of when a line is modified
- QString [prevBlockText](#)
previously accessed line. This is to keep track of when a line is modified
- QString [changedBlockText](#)
current line. This is to keep track of when a line is modified
- QString [prevText](#)
current text. This is to keep track of when the text is modified
- QList< [ItemHandle](#) * > [allItems](#)
all the items represented by the text in this [TextEditor](#)

Friends

- class `TextUndoCommand`
- class `NetworkWindow`
- class `NetworkHandle`
- class `SymbolsTable`
- class `MainWindow`

6.64.1 Detailed Description

This is the window that allows used to construct networks using text, as opposed to graphics, which is done by [GraphicsScene](#). The [TextEditor](#) requires a supporting tool that parses the text and calls the `itemsInserted` or `itemsRemoved` methods. Without a supporting parser tool, the [TextEditor](#) will not do anything.

6.64.2 Member Function Documentation

6.64.2.1 `void Tinkercell::TextEditor::insert (ItemHandle * item)`

insert a text item

Parameters

*ItemHandle** the item

6.64.2.2 `void Tinkercell::TextEditor::insert (const QList< ItemHandle * > & list)`

insert text items

Parameters

QList<ItemHandle>* the items

6.64.2.3 `void Tinkercell::TextEditor::itemsInserted (NetworkHandle *, const QList< ItemHandle * > &) [signal]`

signal that is emitted when items are inserted in this [TextEditor](#).

Parameters

*NetworkHandle**

QList<ItemHandle>* new item handles

6.64.2.4 `void Tinkercell::TextEditor::itemsRemoved (NetworkHandle *, const QList< ItemHandle * > &) [signal]`

signal that is emitted when items are removed from this [TextEditor](#).

Parameters

*NetworkHandle**

QList<ItemHandle>* removed item handles

6.64.2.5 void Tinkercell::TextEditor::lineChanged (TextEditor *, int , const QString &) [signal]

the cursor has moved to a different line

Parameters

int index of the current line

QString current line text

6.64.2.6 void Tinkercell::TextEditor::parse (TextEditor *) [signal]

request to parse the text in the current text editor

Parameters

*TextEditor** editor

6.64.2.7 void Tinkercell::TextEditor::popIn () [virtual, slot]

calls main window's popIn

Returns

void

6.64.2.8 void Tinkercell::TextEditor::popOut () [virtual, slot]

calls main window's popOut

Returns

void

6.64.2.9 void Tinkercell::TextEditor::print (QPrinter * printer) [virtual, slot]

print text

Parameters

QPrinter

6.64.2.10 void Tinkercell::TextEditor::push (QUndoCommand * c)

push a command to the undo/redo stack

Parameters

*QUndoCommand**

6.64.2.11 void Tinkercell::TextEditor::remove (const QList< ItemHandle * > & *handles*)

remove text items

Parameters

QList<ItemHandle>* the items

6.64.2.12 void Tinkercell::TextEditor::remove (ItemHandle * *item*)

remove an item

Parameters

*ItemHandle** the item

6.64.2.13 void Tinkercell::TextEditor::setItems (const QList< ItemHandle * > & *newItems*)

clear existing items and insert new items

Parameters

QList<ItemHandle>* the new items

6.64.2.14 void Tinkercell::TextEditor::textChanged (TextEditor *, const QString &, const QString &, const QString &) [signal]

some text inside this editor has been changed

Parameters

QString old text

QString new text

The documentation for this class was generated from the following files:

- TextEditor.h
- TextEditor.cpp

6.65 Tinkercell::TextGraphicsItem Class Reference

editable text item

```
#include <TextGraphicsItem.h>
```

Public Types

- enum { **Type** = UserType + 8 }
for enabling *dynamic_cast*

Public Member Functions

- virtual [ItemHandle](#) * [handle](#) () const
this text item's handle
- void [setHandle](#) ([ItemHandle](#) *)
set this text item's handle
- [TextGraphicsItem](#) (const [QString](#) &text, [QGraphicsItem](#) *parent=0)
Constructor.
- [TextGraphicsItem](#) ([QGraphicsItem](#) *parent=0)
Constructor.
- [TextGraphicsItem](#) (const [TextGraphicsItem](#) ©)
Copy Constructor.
- virtual [TextGraphicsItem](#) * [clone](#) ()
Clone this item.
- [TextGraphicsItem](#) ([ItemHandle](#) *handle, [QGraphicsItem](#) *parent=0)
Copy Constructor.
- virtual [~TextGraphicsItem](#) ()
Destructor.
- virtual void [paint](#) ([QPainter](#) *painter, const [QStyleOptionGraphicsItem](#) *option, [QWidget](#) *widget)
Paint this text item with or without a border.
- virtual void [showBorder](#) (bool show=true)
whether or not to paint this item with a border
- virtual [QString](#) [text](#) () const
the string painted by this text graphics item. same as toPlainText
- virtual void [setText](#) (const [QString](#) &)
set the string painted by this text graphics item. same as setPlainText
- [QGraphicsItem](#) * [closestItem](#) () const
get the closest item inside the same handle's graphics items list
- int [type](#) () const
for enabling dynamic_cast

Static Public Member Functions

- static [TextGraphicsItem](#) * [cast](#) ([QGraphicsItem](#) *)
cast a graphics item to a text item using qgraphicsitem_cast

Public Attributes

- QPair< QGraphicsItem *, QPointF > [relativePosition](#)
relative position with a target item
- QString [groupID](#)
for identifying which group this item belongs in

Protected Attributes

- QGraphicsRectItem * [boundingRectItem](#)
draws a border around the text item. hide or show using [showBorder\(\)](#)
- [ItemHandle](#) * [itemHandle](#)
the handle in which this item belongs

6.65.1 Detailed Description

editable text item

6.65.2 Constructor & Destructor Documentation

6.65.2.1 Tinkercell::TextGraphicsItem::TextGraphicsItem (const QString & *text*, QGraphicsItem * *parent* = 0)

Constructor.

Parameters

QString *text*

*QGraphicsItem** *parent*

Constructor: sets text edit interaction

6.65.2.2 Tinkercell::TextGraphicsItem::TextGraphicsItem (QGraphicsItem * *parent* = 0)

Constructor.

Parameters

*QGraphicsItem** *parent*

Constructor: sets text edit interaction

6.65.2.3 TinkerCell::TextGraphicsItem::TextGraphicsItem (const TextGraphicsItem & *copy*)

Copy Constructor.

Parameters

*TextGraphicsItem** *copy*

Copy Constructor

6.65.2.4 TinkerCell::TextGraphicsItem::TextGraphicsItem (ItemHandle * *handle*, QGraphicsItem * *parent* = 0)

Copy Constructor.

Parameters

*ItemHandle** *handle* to which this item belongs

*QGraphicsItem** *parent*

Constructor: sets text edit interaction and name of handle

6.65.3 Member Function Documentation

6.65.3.1 TextGraphicsItem * TinkerCell::TextGraphicsItem::cast (QGraphicsItem * *q*) [static]

cast a graphics item to a text item using qgraphicsitem_cast

Parameters

QGraphicsItem *graphics item*

Returns

[TextGraphicsItem](#) this will be 0 if the cast is invalid

6.65.3.2 QGraphicsItem * TinkerCell::TextGraphicsItem::closestItem () const

get the closest item inside the same handle's graphics items list

Returns

[QGraphicsItem](#) *graphics item*

6.65.3.3 void TinkerCell::TextGraphicsItem::setText (const QString & *s*) [virtual]

set the string painted by this text graphics item. same as setPlainText

Parameters

QString

6.65.3.4 QString Tinkercell::TextGraphicsItem::text () const [virtual]

the string painted by this text graphics item. same as toPlainText

Returns

QString

The documentation for this class was generated from the following files:

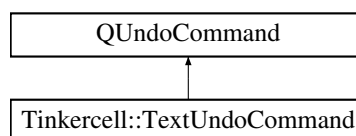
- TextGraphicsItem.h
- TextGraphicsItem.cpp

6.66 Tinkercell::TextUndoCommand Class Reference

this command performs a text change

```
#include <TextEditor.h>
```

Inheritance diagram for Tinkercell::TextUndoCommand:



Public Member Functions

- [TextUndoCommand](#) ([TextEditor](#) *, const QString &, const QString &)
constructor
- void [redo](#) ()
redo the change
- void [undo](#) ()
undo the change

6.66.1 Detailed Description

this command performs a text change

6.66.2 Constructor & Destructor Documentation

6.66.2.1 Tinkercell::TextUndoCommand::TextUndoCommand ([TextEditor](#) * *editor*, const QString & *oldText*, const QString & *newText*)

constructor

Parameters

*TextEditor** editor where change happened

QString new text

The documentation for this class was generated from the following files:

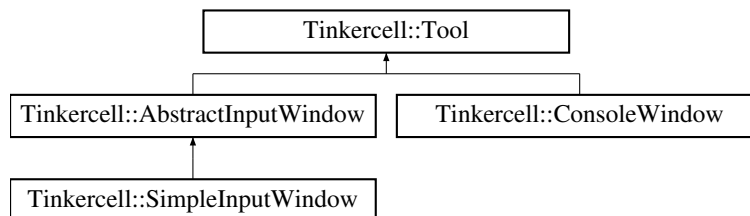
- TextEditor.h
- TextEditor.cpp

6.67 TinkerCell::Tool Class Reference

everything other than the main window is a tool

```
#include <Tool.h>
```

Inheritance diagram for TinkerCell::Tool:

**Public Slots**

- virtual void **select** (int i=0)
what happens when this tool is selected
- virtual void **deselect** (int i=0)
what happens when this tool is deselected
- virtual void **addAction** (const QIcon &, const QString &text=QString(), const QString &tooltip=QString())
add an action that will be displayed in the context menu when specific items with this tool in their tools list are selected
- virtual void **addGraphicsItem** (ToolGraphicsItem *)
add a graphics item that will be displayed on the current scene when specific items with this tool in their tools list are selected

Signals

- void **selected** ()
this tool is selected

- void [deselected](#) ()
this tool is deselected

Public Member Functions

- [Tool](#) ()
constructor
- [~Tool](#) ()
destructor: removes graphicsItem and toolButton is not 0
- [Tool](#) (const QString &Name, const QString &category=QString(), QWidget *parent=0)
constructor
- virtual bool [setMainWindow](#) ([MainWindow](#) *main)
set the main window for this tool
- [ConsoleWindow](#) * [console](#) ()
console window (same as mainWindow->console())
- [GraphicsScene](#) * [currentScene](#) () const
the main window's current scene
- [TextEditor](#) * [currentTextEditor](#) () const
the main window's current text editor
- [NetworkHandle](#) * [currentNetwork](#) () const
the main window's current network
- [NetworkWindow](#) * [currentWindow](#) () const
the main window's current network's current window
- QPair< QList< [ItemHandle](#) * >, QList< QGraphicsItem * > > [getItemsFromFile](#) (const QString &filename)
get the items inside a file. Some tool must implement this function and connect to the getItemsFromFile signal. The Core library does not implement a read file function.

Static Public Member Functions

- static QString [homeDir](#) ()
same as [MainWindow::homeDir](#)
- static QString [tempDir](#) ()
same as [MainWindow::tempDir](#)

Public Attributes

- QString [name](#)
name of this tool
- QString [category](#)
category that this tool belongs in
- QString [description](#)
brief description of this tool
- [MainWindow](#) * [mainWindow](#)
main window for this tool

Protected Slots

- virtual void [actionTriggered](#) (QAction *action)
context menu action triggered

Friends

- class [GraphicsScene](#)
- class [TextEditor](#)
- class [MainWindow](#)
- class [NetworkHandle](#)
- class [ToolGraphicsItem](#)

6.67.1 Detailed Description

everything other than the main window is a tool

6.67.2 Constructor & Destructor Documentation

6.67.2.1 Tinkercell::Tool::Tool (const QString & *Name*, const QString & *category* = *QString()*, QWidget * *parent* = 0)

constructor

Parameters

- QString* *name*
QString *category* (default = empty)
*QWidget** *parent* (default = 0)

6.67.3 Member Function Documentation

6.67.3.1 NetworkHandle * Tinkercell::Tool::currentNetwork () const

the main window's current network

Returns

NetworkHandle* current network handle

6.67.3.2 NetworkWindow * Tinkercell::Tool::currentWindow () const

the main window's current network's current window

Returns

NetworkWindow* current network window

6.67.3.3 QPair< QList< ItemHandle * >, QList< QGraphicsItem * > > Tinkercell::Tool::getItemsFromFile (const QString & filename)

get the items inside a file. Some tool must implement this function and connect to the getItemsFromFile signal. The Core library does not implement a read file function.

Parameters

QString& file that is selected by user

Returns

QPair< QList<ItemHandle*>, QList<QGraphicsItem*> > list of handles and graphics items inside the file
void

The documentation for this class was generated from the following files:

- Tool.h
- Tool.cpp

6.68 Tinkercell::ToolGraphicsItem Class Reference

tools that are drawn on the scene instead of displayed as a window

```
#include <Tool.h>
```

Public Types

- enum { **Type** = UserType + 9 }
for enabling dynamic_cast

Public Member Functions

- [ToolGraphicsItem](#) ([Tool](#) *)
constructor must have an associated [Tool](#)
- virtual void [select](#) ()
this item has been selected
- virtual void [deselect](#) ()
this item has been deselected
- int [type](#) () const
for enabling `dynamic_cast`
- virtual void [visible](#) (bool)
show or hide this graphical tool. The graphical tool may choose whether or not to be visible based on other factors.

Static Public Member Functions

- static [ToolGraphicsItem](#) * [cast](#) ([QGraphicsItem](#) *)
cast a graphics item to a [ToolGraphicsItem](#)

Public Attributes

- [Tool](#) * [tool](#)
main window for this tool

6.68.1 Detailed Description

tools that are drawn on the scene instead of displayed as a window

6.68.2 Member Function Documentation

6.68.2.1 [ToolGraphicsItem](#) * [TinkerCell::ToolGraphicsItem::cast](#) ([QGraphicsItem](#) * *q*) [**static**]

cast a graphics item to a [ToolGraphicsItem](#)

Returns

[ToolGraphicsItem](#)* can be 0 if invalid cast

The documentation for this class was generated from the following files:

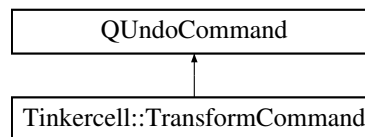
- [Tool.h](#)
- [Tool.cpp](#)

6.69 TinkerCell::TransformCommand Class Reference

this command changes the size, angle, and orientation of an item

```
#include <UndoCommands.h>
```

Inheritance diagram for TinkerCell::TransformCommand:



Public Member Functions

- **TransformCommand** (const QString &name, QGraphicsScene *scene, QGraphicsItem *item, const QPointF &sizechange, qreal anglechange, bool VFlip, bool HFlip)
constructor
- **TransformCommand** (const QString &name, QGraphicsScene *scene, const QList< QGraphicsItem * > &items, const QList< QPointF > &sizechange, const QList< qreal > &anglechange, const QList< bool > &VFlip, const QList< bool > &HFlip)
constructor
- void **redo** ()
- void **undo** ()

6.69.1 Detailed Description

this command changes the size, angle, and orientation of an item

6.69.2 Constructor & Destructor Documentation

6.69.2.1 TinkerCell::TransformCommand::TransformCommand (const QString & name, QGraphicsScene * scene, QGraphicsItem * item, const QPointF & sizechange, qreal anglechange, bool VFlip, bool HFlip)

constructor

Parameters

- QString* name of command
- GraphicsScene** scene where change happened
- GraphicsItem** item that is affected
- PointF* change in size (w,h)
- double* angle change
- boolean* flip vertically
- boolean* flip horizontally

6.69.2.2 Tinkercell::TransformCommand::TransformCommand (const QString & *name*, QGraphicsScene * *scene*, const QList< QGraphicsItem * > & *items*, const QList< QPointF > & *sizechange*, const QList< qreal > & *anglechange*, const QList< bool > & *VFlip*, const QList< bool > & *HFlip*)

constructor

Parameters

QString name of command
*GraphicsScene** scene where change happened
*QList<GGraphicsItem **>& items that are affected
QList<QPointF>& change in size (w,h)
QList<qreal>& angle change
boolean flip vertically (all items)
boolean flip horizontally (all items)

The documentation for this class was generated from the following files:

- UndoCommands.h
- UndoCommands.cpp

6.70 Tinkercell::Unit Class Reference

A unit of measurement.

```
#include <ItemFamily.h>
```

Public Member Functions

- **Unit** (const QString &property, const QString &name)

Public Attributes

- QString **property**
- QString **name**

6.70.1 Detailed Description

A unit of measurement.

The documentation for this class was generated from the following files:

- ItemFamily.h
- ItemFamily.cpp

Index

- ~ConnectionGraphicsItem
 - Tinkercell::ConnectionGraphicsItem, [67](#)
- ~ControlPoint
 - Tinkercell::ConnectionGraphicsItem::ControlPoint, [85](#)
- ~MainWindow
 - Tinkercell::MainWindow, [175](#)
- ~NodeGraphicsItem
 - Tinkercell::NodeGraphicsItem, [230](#)
- AbstractInputWindow
 - Tinkercell::AbstractInputWindow, [33](#)
- AddControlPointCommand
 - Tinkercell::AddControlPointCommand, [34](#)
- AddCurveSegmentCommand
 - Tinkercell::AddCurveSegmentCommand, [37](#)
- addItem
 - Tinkercell::GraphicsScene, [126](#)
- addNode
 - Tinkercell::ConnectionHandle, [77](#)
- AddOptions
 - Tinkercell::SimpleInputWindow, [266](#)
- addParticipant
 - Tinkercell::ConnectionFamily, [60](#)
- addTool
 - Tinkercell::MainWindow, [176](#)
- addToolWindow
 - Tinkercell::MainWindow, [176](#)
- addToViewMenu
 - Tinkercell::MainWindow, [176](#)
- adjustEndPoints
 - Tinkercell::ConnectionGraphicsItem, [67](#)
- allChildren
 - Tinkercell::ItemFamily, [154](#)
 - Tinkercell::ItemHandle, [158](#)
- allGraphicsItems
 - Tinkercell::ItemHandle, [158](#)
- allowMultipleViewModes
 - Tinkercell::MainWindow, [176](#)
- annotations
 - Tinkercell::NetworkHandle, [208](#)
- arrowAt
 - Tinkercell::ConnectionGraphicsItem, [68](#)
- ArrowHeadItem
 - Tinkercell::ArrowHeadItem, [39](#)
- arrowHeads
 - Tinkercell::ConnectionGraphicsItem, [68](#)
- arrowHeadsAsGraphicsItems
 - Tinkercell::ConnectionGraphicsItem, [68](#)
- at
 - Tinkercell::DataTable, [104](#), [105](#)
- autoUnload
 - Tinkercell::CThread, [96](#)
- boundingRect
 - Tinkercell::NodeGraphicsItem::Shape, [262](#)
- C API, [29](#)
- cast
 - Tinkercell::ArrowHeadItem, [40](#)
 - Tinkercell::ConnectionGraphicsItem, [68](#), [69](#)
 - Tinkercell::ConnectionHandle, [78](#)
 - Tinkercell::NodeGraphicsItem, [231](#)
 - Tinkercell::NodeHandle, [236](#)
 - Tinkercell::TextGraphicsItem, [279](#)
 - Tinkercell::ToolGraphicsItem, [285](#)
- centerLocation
 - Tinkercell::ConnectionGraphicsItem, [69](#)
- centerOn
 - Tinkercell::GraphicsScene, [126](#)
- Change2DataCommand
 - Tinkercell::Change2DataCommand, [43](#)
- ChangeBrushAndPenCommand
 - Tinkercell::ChangeBrushAndPenCommand, [45](#)
- ChangeBrushCommand
 - Tinkercell::ChangeBrushCommand, [46](#)
- changeConsoleBgColor
 - Tinkercell::MainWindow, [176](#)
- changeConsoleErrorMsgColor
 - Tinkercell::MainWindow, [177](#)
- changeConsoleMsgColor
 - Tinkercell::MainWindow, [177](#)
- changeConsoleTextColor
 - Tinkercell::MainWindow, [177](#)
- changeData
 - Tinkercell::NetworkHandle, [209](#), [210](#)
- ChangeDataCommand
 - Tinkercell::ChangeDataCommand, [48](#)
- changeEvent

- Tinkercell::NetworkWindow, 220
- ChangeParentCommand
 - Tinkercell::ChangeParentCommand, 49
- ChangePenCommand
 - Tinkercell::ChangePenCommand, 50
- ChangeZCommand
 - Tinkercell::ChangeZCommand, 51, 52
- clear
 - Tinkercell::ConnectionGraphicsItem, 69
 - Tinkercell::NodeGraphicsItem, 231
- clearSelection
 - Tinkercell::GraphicsScene, 127
- clone
 - Tinkercell::ArrowHeadItem, 40
 - Tinkercell::ConnectionGraphicsItem, 69
 - Tinkercell::ConnectionGraphicsItem::ControlPoint, 85
 - Tinkercell::ConnectionHandle, 78
 - Tinkercell::ControlPoint, 88
 - Tinkercell::NodeGraphicsItem, 231
 - Tinkercell::NodeGraphicsItem::ControlPoint, 83
 - Tinkercell::NodeHandle, 236
- cloneGraphicsItem
 - core, 20
- cloneGraphicsItems
 - core, 20
- closeEvent
 - Tinkercell::MainWindow, 177
 - Tinkercell::NetworkWindow, 220
- closestItem
 - Tinkercell::TextGraphicsItem, 279
- colorChanged
 - Tinkercell::GraphicsScene, 127
 - Tinkercell::MainWindow, 177
- columnName
 - Tinkercell::DataTable, 105
- columnNames
 - Tinkercell::DataTable, 105
- columns
 - Tinkercell::DataTable, 106
- CompositeCommand
 - Tinkercell::CompositeCommand, 57
- connectedNodes
 - Tinkercell::NodeGraphicsItem, 231
- ConnectionGraphicsItem
 - Tinkercell::ConnectionGraphicsItem, 67
- ConnectionHandle
 - Tinkercell::ConnectionHandle, 77
- connections
 - Tinkercell::NodeHandle, 236
- connectionsAsGraphicsItems
 - Tinkercell::NodeGraphicsItem, 232
- connectionsDisconnected
 - Tinkercell::NodeGraphicsItem, 232
- connectionsWithArrows
 - Tinkercell::NodeGraphicsItem, 232
- connectionsWithoutArrows
 - Tinkercell::NodeGraphicsItem, 232
- contextMenuEvent
 - Tinkercell::GraphicsScene, 127
- ControlPoint
 - Tinkercell::ControlPoint, 88
- ConvertValue
 - helper, 23–25
- copyItems
 - Tinkercell::GraphicsScene, 127
 - Tinkercell::MainWindow, 178
- copyPoints
 - Tinkercell::ConnectionGraphicsItem, 69
- core
 - cloneGraphicsItem, 20
 - cloneGraphicsItems, 20
 - getGraphicsItem, 20
 - getHandle, 21
 - setHandle, 21
- createScene
 - Tinkercell::NetworkHandle, 210
- createTextEditor
 - Tinkercell::NetworkHandle, 211
- CreateWindow
 - Tinkercell::SimpleInputWindow, 267
- CThread
 - Tinkercell::CThread, 96
- currentNetwork
 - Tinkercell::MainWindow, 178
 - Tinkercell::Tool, 284
- currentScene
 - Tinkercell::MainWindow, 178
 - Tinkercell::NetworkHandle, 211
- currentTextEditor
 - Tinkercell::MainWindow, 178
 - Tinkercell::NetworkHandle, 211
- currentWindow
 - Tinkercell::MainWindow, 178
 - Tinkercell::NetworkHandle, 211
 - Tinkercell::Tool, 284
- dataChanged
 - Tinkercell::MainWindow, 179
 - Tinkercell::NetworkHandle, 211
- depth
 - Tinkercell::ItemHandle, 159
- deselect
 - Tinkercell::GraphicsScene, 128
- dialog
 - Tinkercell::ProcessThread, 240
- disableGrid

- Tinkercell::GraphicsScene, 128
- editors
 - Tinkercell::NetworkHandle, 212
- emptyMatrix
 - helper, 25
- enableGrid
 - Tinkercell::GraphicsScene, 128
- errors
 - Tinkercell::ProcessThread, 241
- escapeSignal
 - Tinkercell::GraphicsScene, 129
 - Tinkercell::MainWindow, 179
- exec
 - Tinkercell::AbstractInputWindow, 33
 - Tinkercell::SimpleInputWindow, 268
- family
 - Tinkercell::ConnectionHandle, 78
 - Tinkercell::NodeHandle, 237
- filesDropped
 - Tinkercell::GraphicsScene, 129
- filesLoaded
 - Tinkercell::MainWindow, 179
- findData
 - Tinkercell::NetworkHandle, 212
- findItem
 - Tinkercell::NetworkHandle, 212, 213
- findValidChildFamilies
 - Tinkercell::ConnectionFamily, 60
 - Tinkercell::ConnectionHandle, 78
- fitAll
 - Tinkercell::GraphicsScene, 129
- fitInView
 - Tinkercell::GraphicsScene, 129
- focusInEvent
 - Tinkercell::NetworkWindow, 220
- fullName
 - Tinkercell::ItemHandle, 159
- functonPointersToMainThread
 - Tinkercell::MainWindow, 179
- getGraphicsItem
 - core, 20
- getHandle
 - core, 21
- getItemsFromFile
 - Tinkercell::MainWindow, 180
 - Tinkercell::Tool, 284
- gridSize
 - Tinkercell::GraphicsScene, 130
- handleFamilyChanged
 - Tinkercell::MainWindow, 180
- Tinkercell::NetworkHandle, 213
- handles
 - Tinkercell::NetworkHandle, 213
- handlesChanged
 - Tinkercell::MainWindow, 181
 - Tinkercell::NetworkHandle, 213
- hasColumn
 - Tinkercell::DataTable, 106
- hasNumericalData
 - Tinkercell::ItemHandle, 159
- hasRow
 - Tinkercell::DataTable, 106
- hasTextData
 - Tinkercell::ItemHandle, 159
- helper
 - ConvertValue, 23–25
 - emptyMatrix, 25
 - pointOnEdge, 25
 - RemoveDisallowedCharactersFromName, 26
- Helper functions and classes, 21
- hideControlPoints
 - Tinkercell::ConnectionGraphicsItem, 69
- historyChanged
 - Tinkercell::MainWindow, 181
 - Tinkercell::NetworkHandle, 214
- historyStack
 - Tinkercell::MainWindow, 181
- historyWidget
 - Tinkercell::MainWindow, 181
- indexOf
 - Tinkercell::ConnectionGraphicsItem, 70
- initializeMenus
 - Tinkercell::MainWindow, 181
- Input and output, 26
- insert
 - Tinkercell::GraphicsScene, 130
 - Tinkercell::TextEditor, 274
- insertColumn
 - Tinkercell::DataTable, 106
- InsertGraphicsCommand
 - Tinkercell::InsertGraphicsCommand, 149
- InsertHandlesCommand
 - Tinkercell::InsertHandlesCommand, 150, 151
- insertRow
 - Tinkercell::DataTable, 107
- isA
 - Tinkercell::ConnectionFamily, 60
 - Tinkercell::ItemHandle, 159
 - Tinkercell::NodeFamily, 225
- isChildOf
 - Tinkercell::ItemHandle, 160
- isModifier
 - Tinkercell::ConnectionGraphicsItem, 70

- isValid
 - TinkerCell::ConnectionGraphicsItem, 70
- isValidSet
 - TinkerCell::ConnectionFamily, 60
- ItemFamily
 - TinkerCell::ItemFamily, 154
- ItemHandle
 - TinkerCell::ItemHandle, 158
- itemsAboutToBeInserted
 - TinkerCell::GraphicsScene, 130
 - TinkerCell::MainWindow, 182
- itemsAboutToBeMoved
 - TinkerCell::GraphicsScene, 131
 - TinkerCell::MainWindow, 182
- itemsAboutToBeRemoved
 - TinkerCell::GraphicsScene, 131
 - TinkerCell::MainWindow, 182
- itemsDropped
 - TinkerCell::MainWindow, 183
- itemsInserted
 - TinkerCell::GraphicsScene, 131
 - TinkerCell::MainWindow, 183
 - TinkerCell::TextEditor, 274
- itemsInsertedSlot
 - TinkerCell::MainWindow, 183
- itemsMoved
 - TinkerCell::GraphicsScene, 132
 - TinkerCell::MainWindow, 184
- itemsRemoved
 - TinkerCell::GraphicsScene, 132
 - TinkerCell::MainWindow, 184
 - TinkerCell::TextEditor, 274
- itemsRemovedSlot
 - TinkerCell::MainWindow, 185
- itemsRenamed
 - TinkerCell::MainWindow, 185
 - TinkerCell::NetworkHandle, 214
- itemsSelected
 - TinkerCell::GraphicsScene, 132
 - TinkerCell::MainWindow, 185
- keyPressed
 - TinkerCell::GraphicsScene, 133
 - TinkerCell::MainWindow, 186
- keyPressEvent
 - TinkerCell::GraphicsScene, 133
- keyReleased
 - TinkerCell::GraphicsScene, 133
 - TinkerCell::MainWindow, 186
- keyReleaseEvent
 - TinkerCell::GraphicsScene, 133
- lastPoint
 - TinkerCell::GraphicsScene, 134
- lastScreenPoint
 - TinkerCell::GraphicsScene, 134
- library
 - TinkerCell::CThread, 97
- lineChanged
 - TinkerCell::MainWindow, 186
 - TinkerCell::TextEditor, 274
- loadDefaultPlugins
 - TinkerCell::MainWindow, 186
- loadDynamicLibrary
 - TinkerCell::MainWindow, 187
- loadFiles
 - TinkerCell::MainWindow, 187
- loadLibrary
 - TinkerCell::CThread, 97
- loadNetwork
 - TinkerCell::MainWindow, 187
- MainWindow
 - TinkerCell::MainWindow, 175
- makeUnique
 - TinkerCell::NetworkHandle, 214, 215
- mapToWidget
 - TinkerCell::GraphicsScene, 135
- message
 - TinkerCell::ConsoleWindow, 82
- modifierArrowAt
 - TinkerCell::ConnectionGraphicsItem, 70
- modifierArrowHeads
 - TinkerCell::ConnectionGraphicsItem, 70
- mouseDoubleClicked
 - TinkerCell::GraphicsScene, 135
 - TinkerCell::MainWindow, 187
- mouseDoubleClickEvent
 - TinkerCell::GraphicsScene, 135
- mouseDragged
 - TinkerCell::GraphicsScene, 136
 - TinkerCell::MainWindow, 188
- mouseMoved
 - TinkerCell::GraphicsScene, 136
 - TinkerCell::MainWindow, 188
- mouseMoveEvent
 - TinkerCell::GraphicsScene, 136
- mouseOnTopOf
 - TinkerCell::GraphicsScene, 137
 - TinkerCell::MainWindow, 188
- mousePressed
 - TinkerCell::GraphicsScene, 137
 - TinkerCell::MainWindow, 189
- mousePressEvent
 - TinkerCell::GraphicsScene, 137
- mouseReleased
 - TinkerCell::GraphicsScene, 138
 - TinkerCell::MainWindow, 189

- mouseReleaseEvent
 - TinkerCell::GraphicsScene, 138
- move
 - TinkerCell::GraphicsScene, 139
- MoveCommand
 - TinkerCell::MoveCommand, 197, 198
- moving
 - TinkerCell::GraphicsScene, 139
- MultithreadedSliderWidget
 - TinkerCell::MultithreadedSliderWidget, 201
- negative
 - TinkerCell::NodeGraphicsItem::Shape, 263
- networkClosed
 - TinkerCell::MainWindow, 189
 - TinkerCell::NetworkWindow, 221
- networkClosing
 - TinkerCell::MainWindow, 190
 - TinkerCell::NetworkWindow, 221
- networkLoaded
 - TinkerCell::MainWindow, 190
- networkOpened
 - TinkerCell::MainWindow, 190
- networks
 - TinkerCell::MainWindow, 190
- networkSaved
 - TinkerCell::MainWindow, 191
- newScene
 - TinkerCell::NetworkWindow, 221
- newTextEditor
 - TinkerCell::NetworkWindow, 221
- nodeAt
 - TinkerCell::ConnectionGraphicsItem, 71
- NodeFamily
 - TinkerCell::NodeFamily, 224
- NodeGraphicsItem
 - TinkerCell::NodeGraphicsItem, 230
- NodeHandle
 - TinkerCell::NodeHandle, 236
- nodeItem
 - TinkerCell::NodeGraphicsItem::Shape, 263
- nodes
 - TinkerCell::ConnectionGraphicsItem, 71
 - TinkerCell::ConnectionHandle, 79
- nodesAsGraphicsItems
 - TinkerCell::ConnectionGraphicsItem, 71
- nodesDisconnected
 - TinkerCell::ConnectionGraphicsItem, 71
- nodesIn
 - TinkerCell::ConnectionHandle, 79
- nodesOut
 - TinkerCell::ConnectionHandle, 79
- nodesWithArrows
 - TinkerCell::ConnectionGraphicsItem, 72
- nodesWithoutArrows
 - TinkerCell::ConnectionGraphicsItem, 72
- normalize
 - TinkerCell::NodeGraphicsItem, 232
- numberOfIdenticalNodesFamilies
 - TinkerCell::ConnectionFamily, 61
- numericalData
 - TinkerCell::ItemHandle, 160, 161
- numericalDataNames
 - TinkerCell::ItemHandle, 161
- numericalDataTable
 - TinkerCell::ItemHandle, 161
- operator()
 - TinkerCell::DataTable, 107–109
- operator=
 - TinkerCell::ConnectionGraphicsItem, 72
 - TinkerCell::ConnectionGraphicsItem::ControlPoint, 85
 - TinkerCell::NodeGraphicsItem, 233
 - TinkerCell::NodeGraphicsItem::ControlPoint, 83
 - TinkerCell::NodeGraphicsItem::Shape, 262
- operator==
 - TinkerCell::DataTable, 109
- output
 - TinkerCell::ProcessThread, 241
- paint
 - TinkerCell::ArrowHeadItem, 40
 - TinkerCell::ControlPoint, 88
 - TinkerCell::NodeGraphicsItem::ControlPoint, 83
- parentHandleChanged
 - TinkerCell::MainWindow, 191
 - TinkerCell::NetworkHandle, 215
- parentItemChanged
 - TinkerCell::GraphicsScene, 140
 - TinkerCell::MainWindow, 191
- parentOfFamily
 - TinkerCell::ItemHandle, 161
- parse
 - TinkerCell::MainWindow, 191
 - TinkerCell::TextEditor, 275
- parseMath
 - TinkerCell::NetworkHandle, 215
- participantFamily
 - TinkerCell::ConnectionFamily, 61
- participantRoles
 - TinkerCell::ConnectionFamily, 61
- participantTypes
 - TinkerCell::ConnectionFamily, 61
- pen
 - TinkerCell::ConnectionGraphicsItem, 72

- Plotting, 29
- pointOnEdge
 - helper, 25
- polygon
 - TinkerCell::NodeGraphicsItem, 233
- popIn
 - TinkerCell::GraphicsScene, 140
 - TinkerCell::NetworkWindow, 221
 - TinkerCell::TextEditor, 275
- popOut
 - TinkerCell::GraphicsScene, 140
 - TinkerCell::NetworkWindow, 222
 - TinkerCell::TextEditor, 275
- populateContextMenu
 - TinkerCell::GraphicsScene, 140
- prepareNetworkForSaving
 - TinkerCell::MainWindow, 192
- print
 - TinkerCell::GraphicsScene, 141
 - TinkerCell::MainWindow, 192
 - TinkerCell::TextEditor, 275
- printToFile
 - TinkerCell::MainWindow, 192
- ProcessThread
 - TinkerCell::ProcessThread, 240
- push
 - TinkerCell::TextEditor, 275
- QUndoCommand, 241
- readSettings
 - TinkerCell::MainWindow, 192
- rect
 - TinkerCell::ControlPoint, 88
- redo
 - TinkerCell::AddControlPointCommand, 35
 - TinkerCell::AddCurveSegmentCommand, 37
 - TinkerCell::RemoveControlPointCommand, 244
 - TinkerCell::RemoveCurveSegmentCommand, 246
- refresh
 - TinkerCell::ConnectionGraphicsItem, 72
 - TinkerCell::NodeGraphicsItem, 233
 - TinkerCell::NodeGraphicsItem::Shape, 262
- refreshAllConnectionIn
 - TinkerCell::MoveCommand, 198
- remove
 - TinkerCell::GraphicsScene, 141
 - TinkerCell::TextEditor, 275, 276
- removeColumn
 - TinkerCell::DataTable, 110
- RemoveControlPointCommand
 - TinkerCell::RemoveControlPointCommand, 243
- RemoveCurveSegmentCommand
 - TinkerCell::RemoveCurveSegmentCommand, 246
- RemoveDisallowedCharactersFromName
 - helper, 26
- RemoveGraphicsCommand
 - TinkerCell::RemoveGraphicsCommand, 248
- RemoveHandlesCommand
 - TinkerCell::RemoveHandlesCommand, 249
- removeRow
 - TinkerCell::DataTable, 110
- RenameCommand
 - TinkerCell::RenameCommand, 251–253
- ReplaceConnectedNodeCommand
 - TinkerCell::ReplaceConnectedNodeCommand, 254
- replaceNode
 - TinkerCell::ConnectionGraphicsItem, 73
- replaceNodeAt
 - TinkerCell::ConnectionGraphicsItem, 73
- ReplaceNodeGraphicsCommand
 - TinkerCell::ReplaceNodeGraphicsCommand, 255
- resetBrush
 - TinkerCell::NodeGraphicsItem, 233
- resetPen
 - TinkerCell::NodeGraphicsItem, 233
- resetToDefaults
 - TinkerCell::NodeGraphicsItem, 233
- resize
 - TinkerCell::DataTable, 111
- resizeEvent
 - TinkerCell::NetworkWindow, 222
- ReverseUndoCommand
 - TinkerCell::ReverseUndoCommand, 257
- root
 - TinkerCell::ItemHandle, 162
- rowName
 - TinkerCell::DataTable, 111
- rowNames
 - TinkerCell::DataTable, 111
- rows
 - TinkerCell::DataTable, 112
- saveNetwork
 - TinkerCell::MainWindow, 192
- saveSettings
 - TinkerCell::MainWindow, 192
- sceneRightClick
 - TinkerCell::GraphicsScene, 141
 - TinkerCell::MainWindow, 193
- scenes

- Tinkercell::NetworkHandle, 216
- select
 - Tinkercell::GraphicsScene, 142
- selected
 - Tinkercell::GraphicsScene, 142
- selectedRect
 - Tinkercell::GraphicsScene, 142
- setAlpha
 - Tinkercell::NodeGraphicsItem, 233
- setArg
 - Tinkercell::CThread, 97
- setAsCurrentWindow
 - Tinkercell::NetworkWindow, 222
- setAutoUnload
 - Tinkercell::CThread, 97
- setBrush
 - Tinkercell::GraphicsScene, 143
- setBrushAndPen
 - Tinkercell::GraphicsScene, 143
- setCharFunction
 - Tinkercell::CThread, 98
- setColumnName
 - Tinkercell::DataTable, 112
- setColumnNames
 - Tinkercell::DataTable, 112
- setControlPointsVisible
 - Tinkercell::ConnectionGraphicsItem, 73
- setCursor
 - Tinkercell::MainWindow, 193
- setDoubleFunction
 - Tinkercell::CThread, 98
- setFamily
 - Tinkercell::ConnectionHandle, 79
 - Tinkercell::NodeHandle, 237
- setFileName
 - Tinkercell::NetworkWindow, 222
- setFunction
 - Tinkercell::CThread, 98
- setGridSize
 - Tinkercell::GraphicsScene, 143
- setHandle
 - core, 21
- setItems
 - Tinkercell::TextEditor, 276
- setLibrary
 - Tinkercell::CThread, 99
- setMatrixFunction
 - Tinkercell::CThread, 99
- setModelValues
 - Tinkercell::NetworkHandle, 216, 217
- setParent
 - Tinkercell::ItemHandle, 162
- setParentItem
 - Tinkercell::GraphicsScene, 144
- setPath
 - Tinkercell::ConnectionGraphicsItem, 74
- setPen
 - Tinkercell::ConnectionGraphicsItem, 74
 - Tinkercell::GraphicsScene, 144
- setRect
 - Tinkercell::ControlPoint, 88
- setRowName
 - Tinkercell::DataTable, 112
- setRowNames
 - Tinkercell::DataTable, 113
- setSliders
 - Tinkercell::MultithreadedSliderWidget, 201
- setText
 - Tinkercell::TextGraphicsItem, 279
- setTitle
 - Tinkercell::CThread, 99
- setupFunctionPointers
 - Tinkercell::MainWindow, 193
- setupFunctionPointersSlot
 - Tinkercell::MainWindow, 193
- setupNewThread
 - Tinkercell::MainWindow, 194
- setVisibleSliders
 - Tinkercell::MultithreadedSliderWidget, 201, 202
- setVoidFunction
 - Tinkercell::CThread, 99
- setWindowTitle
 - Tinkercell::NetworkHandle, 217
 - Tinkercell::NetworkWindow, 222
- Shape
 - Tinkercell::NodeGraphicsItem::Shape, 261
- shape
 - Tinkercell::ConnectionGraphicsItem, 74
 - Tinkercell::NodeGraphicsItem, 234
 - Tinkercell::NodeGraphicsItem::Shape, 262
- showControlPoints
 - Tinkercell::ConnectionGraphicsItem, 74
- showProgress
 - Tinkercell::CThread, 99
- showScene
 - Tinkercell::NetworkHandle, 217
- showTextEditor
 - Tinkercell::NetworkHandle, 217
- SimpleInputWindow
 - Tinkercell::SimpleInputWindow, 265, 266
- slopeAtPoint
 - Tinkercell::ConnectionGraphicsItem, 74
- snapToGrid
 - Tinkercell::GraphicsScene, 144
- swapColumns
 - Tinkercell::DataTable, 113
- swapRows

- Tinkercell::DataTable, 113, 114
- SymbolsTable
 - Tinkercell::SymbolsTable, 270
- symbolsTable
 - Tinkercell::NetworkHandle, 218
- synonyms
 - Tinkercell::ConnectionFamily, 61
- text
 - Tinkercell::TextGraphicsItem, 279
- textChanged
 - Tinkercell::MainWindow, 194
 - Tinkercell::TextEditor, 276
- textData
 - Tinkercell::ItemHandle, 162, 163
- textDataNames
 - Tinkercell::ItemHandle, 163
- textDataTable
 - Tinkercell::ItemHandle, 163
- TextGraphicsItem
 - Tinkercell::TextGraphicsItem, 278, 279
- TextUndoCommand
 - Tinkercell::TextUndoCommand, 280
- TinkerCell Core classes, 17
- TinkerCell plug-ins, 30
- Tinkercell::AbstractInputWindow, 31
 - AbstractInputWindow, 33
 - exec, 33
- Tinkercell::AddControlPointCommand, 33
 - AddControlPointCommand, 34
 - redo, 35
 - undo, 35
- Tinkercell::AddCurveSegmentCommand, 35
 - AddCurveSegmentCommand, 37
 - redo, 37
 - undo, 37
- Tinkercell::ArrowHeadItem, 38
 - ArrowHeadItem, 39
 - cast, 40
 - clone, 40
 - paint, 40
- Tinkercell::AssignHandleCommand, 41
- Tinkercell::C_API_Slots, 41
- Tinkercell::Change2DataCommand, 42
 - Change2DataCommand, 43
- Tinkercell::ChangeBrushAndPenCommand, 44
 - ChangeBrushAndPenCommand, 45
- Tinkercell::ChangeBrushCommand, 45
 - ChangeBrushCommand, 46
- Tinkercell::ChangeDataCommand, 47
 - ChangeDataCommand, 48
- Tinkercell::ChangeParentCommand, 48
 - ChangeParentCommand, 49
- Tinkercell::ChangePenCommand, 50
 - ChangePenCommand, 50
- Tinkercell::ChangeZCommand, 51
 - ChangeZCommand, 51, 52
- Tinkercell::CodeEditor, 52
- Tinkercell::CommandTextEdit, 53
- Tinkercell::CompositeCommand, 56
 - CompositeCommand, 57
- Tinkercell::ConnectionFamily, 57
 - addParticipant, 60
 - findValidChildFamilies, 60
 - isA, 60
 - isValidSet, 60
 - numberOfIdenticalNodesFamilies, 61
 - participantFamily, 61
 - participantRoles, 61
 - participantTypes, 61
 - synonyms, 61
- Tinkercell::ConnectionGraphicsItem, 62
 - ~ConnectionGraphicsItem, 67
 - adjustEndPoints, 67
 - arrowAt, 68
 - arrowHeads, 68
 - arrowHeadsAsGraphicsItems, 68
 - cast, 68, 69
 - centerLocation, 69
 - clear, 69
 - clone, 69
 - ConnectionGraphicsItem, 67
 - copyPoints, 69
 - hideControlPoints, 69
 - indexOf, 70
 - isModifier, 70
 - isValid, 70
 - modifierArrowAt, 70
 - modifierArrowHeads, 70
 - nodeAt, 71
 - nodes, 71
 - nodesAsGraphicsItems, 71
 - nodesDisconnected, 71
 - nodesWithArrows, 72
 - nodesWithoutArrows, 72
 - operator=, 72
 - pen, 72
 - refresh, 72
 - replaceNode, 73
 - replaceNodeAt, 73
 - setControlPointsVisible, 73
 - setPath, 74
 - setPen, 74
 - shape, 74
 - showControlPoints, 74
 - slopeAtPoint, 74
 - topLevelConnectionItem, 75

- Tinkercell::ConnectionGraphicsItem::ControlPoint, 84
 - ~ControlPoint, 85
 - clone, 85
 - operator=, 85
- Tinkercell::ConnectionGraphicsItem::CurveSegment, 100
- Tinkercell::ConnectionHandle, 75
 - addNode, 77
 - cast, 78
 - clone, 78
 - ConnectionHandle, 77
 - family, 78
 - findValidChildFamilies, 78
 - nodes, 79
 - nodesIn, 79
 - nodesOut, 79
 - setFamily, 79
- Tinkercell::ConsoleWindow, 80
 - message, 82
- Tinkercell::ControlPoint, 86
 - clone, 88
 - ControlPoint, 88
 - paint, 88
 - rect, 88
 - setRect, 88
- Tinkercell::Core_FtoS, 89
- Tinkercell::CThread, 92
 - autoUnload, 96
 - CThread, 96
 - library, 97
 - loadLibrary, 97
 - setArg, 97
 - setAutoUnload, 97
 - setCharFunction, 98
 - setDoubleFunction, 98
 - setFunction, 98
 - setLibrary, 99
 - setMatrixFunction, 99
 - setTitle, 99
 - setVoidFunction, 99
 - showProgress, 99
- Tinkercell::DataTable, 100
 - at, 104, 105
 - columnName, 105
 - columnNames, 105
 - columns, 106
 - hasColumn, 106
 - hasRow, 106
 - insertColumn, 106
 - insertRow, 107
 - operator(), 107–109
 - operator==, 109
 - removeColumn, 110
 - removeRow, 110
 - resize, 111
 - rowName, 111
 - rowNames, 111
 - rows, 112
 - setColumnName, 112
 - setColumnNames, 112
 - setRowName, 112
 - setRowNames, 113
 - swapColumns, 113
 - swapRows, 113, 114
 - transpose, 114
 - value, 114, 115
- Tinkercell::GraphicsScene, 116
 - addItem, 126
 - centerOn, 126
 - clearSelection, 127
 - colorChanged, 127
 - contextMenuEvent, 127
 - copyItems, 127
 - deselect, 128
 - disableGrid, 128
 - enableGrid, 128
 - escapeSignal, 129
 - filesDropped, 129
 - fitAll, 129
 - fitInView, 129
 - gridSize, 130
 - insert, 130
 - itemsAboutToBeInserted, 130
 - itemsAboutToBeMoved, 131
 - itemsAboutToBeRemoved, 131
 - itemsInserted, 131
 - itemsMoved, 132
 - itemsRemoved, 132
 - itemsSelected, 132
 - keyPressed, 133
 - keyPressEvent, 133
 - keyReleased, 133
 - keyReleaseEvent, 133
 - lastPoint, 134
 - lastScreenPoint, 134
 - mapToWidget, 135
 - mouseDoubleClicked, 135
 - mouseDoubleClickEvent, 135
 - mouseDragged, 136
 - mouseMoved, 136
 - mouseMoveEvent, 136
 - mouseOnTopOf, 137
 - mousePressed, 137
 - mousePressEvent, 137
 - mouseReleased, 138
 - mouseReleaseEvent, 138
 - move, 139

- moving, 139
- parentItemChanged, 140
- popIn, 140
- popOut, 140
- populateContextMenu, 140
- print, 141
- remove, 141
- sceneRightClick, 141
- select, 142
- selected, 142
- selectedRect, 142
- setBrush, 143
- setBrushAndPen, 143
- setGridSize, 143
- setParentItem, 144
- setPen, 144
- snapToGrid, 144
- transform, 145
- visibleRegion, 145
- zoom, 145
- zoomIn, 146
- zoomOut, 146
- ZValue, 146
- Tinkercell::GraphicsView, 147
- Tinkercell::HistoryWindow, 148
- Tinkercell::InsertGraphicsCommand, 148
 - InsertGraphicsCommand, 149
- Tinkercell::InsertHandlesCommand, 150
 - InsertHandlesCommand, 150, 151
- Tinkercell::ItemData, 151
- Tinkercell::ItemFamily, 152
 - allChildren, 154
 - ItemFamily, 154
- Tinkercell::ItemHandle, 155
 - allChildren, 158
 - allGraphicsItems, 158
 - depth, 159
 - fullName, 159
 - hasNumericalData, 159
 - hasTextData, 159
 - isA, 159
 - isChildOf, 160
 - ItemHandle, 158
 - numericalData, 160, 161
 - numericalDataNames, 161
 - numericalDataTable, 161
 - parentOfFamily, 161
 - root, 162
 - setParent, 162
 - textData, 162, 163
 - textDataNames, 163
 - textDataTable, 163
- Tinkercell::LineNumberArea, 164
- Tinkercell::MainWindow, 164
 - ~MainWindow, 175
 - addTool, 176
 - addToolWindow, 176
 - addToViewMenu, 176
 - allowMultipleViewModes, 176
 - changeConsoleBgColor, 176
 - changeConsoleErrorMsgColor, 177
 - changeConsoleMsgColor, 177
 - changeConsoleTextColor, 177
 - closeEvent, 177
 - colorChanged, 177
 - copyItems, 178
 - currentNetwork, 178
 - currentScene, 178
 - currentTextEditor, 178
 - currentWindow, 178
 - dataChanged, 179
 - escapeSignal, 179
 - filesLoaded, 179
 - functioNPointersToMainThread, 179
 - getItemsFromFile, 180
 - handleFamilyChanged, 180
 - handlesChanged, 181
 - historyChanged, 181
 - historyStack, 181
 - historyWidget, 181
 - initializeMenus, 181
 - itemsAboutToBeInserted, 182
 - itemsAboutToBeMoved, 182
 - itemsAboutToBeRemoved, 182
 - itemsDropped, 183
 - itemsInserted, 183
 - itemsInsertedSlot, 183
 - itemsMoved, 184
 - itemsRemoved, 184
 - itemsRemovedSlot, 185
 - itemsRenamed, 185
 - itemsSelected, 185
 - keyPressed, 186
 - keyReleased, 186
 - lineChanged, 186
 - loadDefaultPlugins, 186
 - loadDynamicLibrary, 187
 - loadFiles, 187
 - loadNetwork, 187
 - MainWindow, 175
 - mouseDoubleClicked, 187
 - mouseDragged, 188
 - mouseMoved, 188
 - mouseOnTopOf, 188
 - mousePressed, 189
 - mouseReleased, 189
 - networkClosed, 189
 - networkClosing, 190

- networkLoaded, 190
- networkOpened, 190
- networks, 190
- networkSaved, 191
- parentHandleChanged, 191
- parentItemChanged, 191
- parse, 191
- prepareNetworkForSaving, 192
- print, 192
- printToFile, 192
- readSettings, 192
- saveNetwork, 192
- saveSettings, 192
- sceneRightClick, 193
- setCursor, 193
- setupFunctionPointers, 193
- setupFunctionPointersSlot, 193
- setupNewThread, 194
- textChanged, 194
- tool, 194
- toolAboutToBeLoaded, 194
- toolLoaded, 195
- tools, 195
- windowChanged, 195
- Tinkercell::MergeHandlesCommand, 196
- Tinkercell::MoveCommand, 196
 - MoveCommand, 197, 198
 - refreshAllConnectionIn, 198
- Tinkercell::MultithreadedSliderWidget, 198
 - MultithreadedSliderWidget, 201
 - setSliders, 201
 - setVisibleSliders, 201, 202
- Tinkercell::NetworkHandle, 202
 - annotations, 208
 - changeData, 209, 210
 - createScene, 210
 - createTextEditor, 211
 - currentScene, 211
 - currentTextEditor, 211
 - currentWindow, 211
 - dataChanged, 211
 - editors, 212
 - findData, 212
 - findItem, 212, 213
 - handleFamilyChanged, 213
 - handles, 213
 - handlesChanged, 213
 - historyChanged, 214
 - itemsRenamed, 214
 - makeUnique, 214, 215
 - parentHandleChanged, 215
 - parseMath, 215
 - scenes, 216
 - setModelValues, 216, 217
 - setWindowTitle, 217
 - showScene, 217
 - showTextEditor, 217
 - symbolsTable, 218
 - updateSymbolsTable, 217, 218
 - windowTitle, 218
- Tinkercell::NetworkWindow, 218
 - changeEvent, 220
 - closeEvent, 220
 - focusInEvent, 220
 - networkClosed, 221
 - networkClosing, 221
 - newScene, 221
 - newTextEditor, 221
 - popIn, 221
 - popOut, 222
 - resizeEvent, 222
 - setAsCurrentWindow, 222
 - setFileName, 222
 - setWindowTitle, 222
- Tinkercell::NodeFamily, 223
 - isA, 225
 - NodeFamily, 224
- Tinkercell::NodeGraphicsItem, 225
 - ~NodeGraphicsItem, 230
 - cast, 231
 - clear, 231
 - clone, 231
 - connectedNodes, 231
 - connectionsAsGraphicsItems, 232
 - connectionsDisconnected, 232
 - connectionsWithArrows, 232
 - connectionsWithoutArrows, 232
 - NodeGraphicsItem, 230
 - normalize, 232
 - operator=, 233
 - polygon, 233
 - refresh, 233
 - resetBrush, 233
 - resetPen, 233
 - resetToDefaults, 233
 - setAlpha, 233
 - shape, 234
 - topLevelNodeItem, 234
- Tinkercell::NodeGraphicsItem::ControlPoint, 82
 - clone, 83
 - operator=, 83
 - paint, 83
- Tinkercell::NodeGraphicsItem::Shape, 260
 - boundingRect, 262
 - negative, 263
 - nodeItem, 263
 - operator=, 262
 - refresh, 262

- Shape, 261
- shape, 262
- Tinkercell::NodeHandle, 234
 - cast, 236
 - clone, 236
 - connections, 236
 - family, 237
 - NodeHandle, 236
 - setFamily, 237
- Tinkercell::PopupListWidgetDelegate, 237
- Tinkercell::PopupListWidgetDelegateDialog, 238
- Tinkercell::ProcessThread, 239
 - dialog, 240
 - errors, 241
 - output, 241
 - ProcessThread, 240
- Tinkercell::RemoveControlPointCommand, 242
 - redo, 244
 - RemoveControlPointCommand, 243
 - undo, 244
- Tinkercell::RemoveCurveSegmentCommand, 244
 - redo, 246
 - RemoveCurveSegmentCommand, 246
 - undo, 246
- Tinkercell::RemoveGraphicsCommand, 247
 - RemoveGraphicsCommand, 248
- Tinkercell::RemoveHandlesCommand, 248
 - RemoveHandlesCommand, 249
- Tinkercell::RenameCommand, 250
 - RenameCommand, 251–253
- Tinkercell::ReplaceConnectedNodeCommand, 254
 - ReplaceConnectedNodeCommand, 254
- Tinkercell::ReplaceNodeGraphicsCommand, 255
 - ReplaceNodeGraphicsCommand, 255
- Tinkercell::ReverseUndoCommand, 256
 - ReverseUndoCommand, 257
- Tinkercell::SetGraphicsSceneVisibilityCommand, 257
- Tinkercell::SetHandleFamilyCommand, 258
- Tinkercell::SetParentHandleCommand, 259
- Tinkercell::SimpleInputWindow, 263
 - AddOptions, 266
 - CreateWindow, 267
 - exec, 268
 - SimpleInputWindow, 265, 266
- Tinkercell::SymbolsTable, 268
 - SymbolsTable, 270
- Tinkercell::TextEditor, 270
 - insert, 274
 - itemsInserted, 274
 - itemsRemoved, 274
 - lineChanged, 274
 - parse, 275
 - popIn, 275
 - popOut, 275
 - print, 275
 - push, 275
 - remove, 275, 276
 - setItems, 276
 - textChanged, 276
- Tinkercell::TextGraphicsItem, 276
 - cast, 279
 - closestItem, 279
 - setText, 279
 - text, 279
 - TextGraphicsItem, 278, 279
- Tinkercell::TextUndoCommand, 280
 - TextUndoCommand, 280
- Tinkercell::Tool, 281
 - currentNetwork, 284
 - currentWindow, 284
 - getItemsFromFile, 284
 - Tool, 283
- Tinkercell::ToolGraphicsItem, 284
 - cast, 285
- Tinkercell::TransformCommand, 286
 - TransformCommand, 286
- Tinkercell::Unit, 287
- Tool
 - Tinkercell::Tool, 283
- tool
 - Tinkercell::MainWindow, 194
- toolAboutToBeLoaded
 - Tinkercell::MainWindow, 194
- toolLoaded
 - Tinkercell::MainWindow, 195
- tools
 - Tinkercell::MainWindow, 195
- topLevelConnectionItem
 - Tinkercell::ConnectionGraphicsItem, 75
- topLevelNodeItem
 - Tinkercell::NodeGraphicsItem, 234
- transform
 - Tinkercell::GraphicsScene, 145
- TransformCommand
 - Tinkercell::TransformCommand, 286
- transpose
 - Tinkercell::DataTable, 114
- undo
 - Tinkercell::AddControlPointCommand, 35
 - Tinkercell::AddCurveSegmentCommand, 37
 - Tinkercell::RemoveControlPointCommand, 244
 - Tinkercell::RemoveCurveSegmentCommand, 246
- Undo commands, 27
- updateSymbolsTable

- TinkerCell::NetworkHandle, [217](#), [218](#)
- value
 - TinkerCell::DataTable, [114](#), [115](#)
- visibleRegion
 - TinkerCell::GraphicsScene, [145](#)
- windowChanged
 - TinkerCell::MainWindow, [195](#)
- windowTitle
 - TinkerCell::NetworkHandle, [218](#)
- zoom
 - TinkerCell::GraphicsScene, [145](#)
- zoomIn
 - TinkerCell::GraphicsScene, [146](#)
- zoomOut
 - TinkerCell::GraphicsScene, [146](#)
- ZValue
 - TinkerCell::GraphicsScene, [146](#)