

Object Graphics Library 3.0

Julian Smart

September 1998

Contents

Introduction	1
File structure.....	1
OGLEdit: a sample OGL application	3
OGLEdit files	3
How OGLEdit works.....	4
Possible enhancements	4
Class reference	6
wxOGLConstraint	6
wxBitmapShape.....	8
wxDiagram	9
wxDrawnShape	14
wxCircleShape.....	19
wxCompositeShape	19
wxDividedShape	22
wxDivisionShape	22
wxEllipseShape	26
wxLineShape	27
wxPolygonShape	33
wxRectangleShape.....	34
wxPseudoMetaFile	35
wxShape	35
wxShapeCanvas.....	52
wxShapeEvtHandler	57
wxTextShape.....	62
Functions.....	62
Topic overviews.....	63
OGL overview.....	63
wxDividedShape overview	63
wxCompositeShape overview	65
Bugs.....	66
Change log	67
Index	68

Introduction

Object Graphics Library (OGL) is a C++ library supporting the creation and manipulation of simple and complex graphic images on a canvas.

It can be found in the directory `utils/ogl/src` in the wxWindows distribution. The file `ogl.h` must be included to make use of the library.

Please see *OGL overview* (p. 63) for a general description how the object library works. For details, please see the *class reference* (p. 6).

File structure

These are the files that comprise the OGL library.

basic.h Header for basic objects such as `wxShape` and `wxRectangleShape`.

basic.cpp Basic objects implementation (1).

basic2.cpp Basic objects implementation (2).

bmpshape.h `wxBitmapShape` class header.

bmpshape.cpp `wxBitmapShape` implementation.

canvas.h `wxShapeCanvas` class header.

canvas.cpp `wxShapeCanvas` class implementation.

composit.h Composite object class header.

composit.cpp Composite object class implementation.

constrnt.h Constraint classes header.

constrnt.cpp Constraint classes implementation.

divided.h Divided object class header.

divided.cpp Divided object class implementation.

drawn.h Drawn (metafile) object class header.

drawn.cpp Drawn (metafile) object class implementation.

graphics.h Main include file.

lines.h `wxLineShape` class header.

lines.cpp `wxLineShape` class implementation.

misc.h Miscellaneous graphics functions header.

misc.cpp Miscellaneous graphics functions implementation.

ogldiag.h wxDiagram class header.

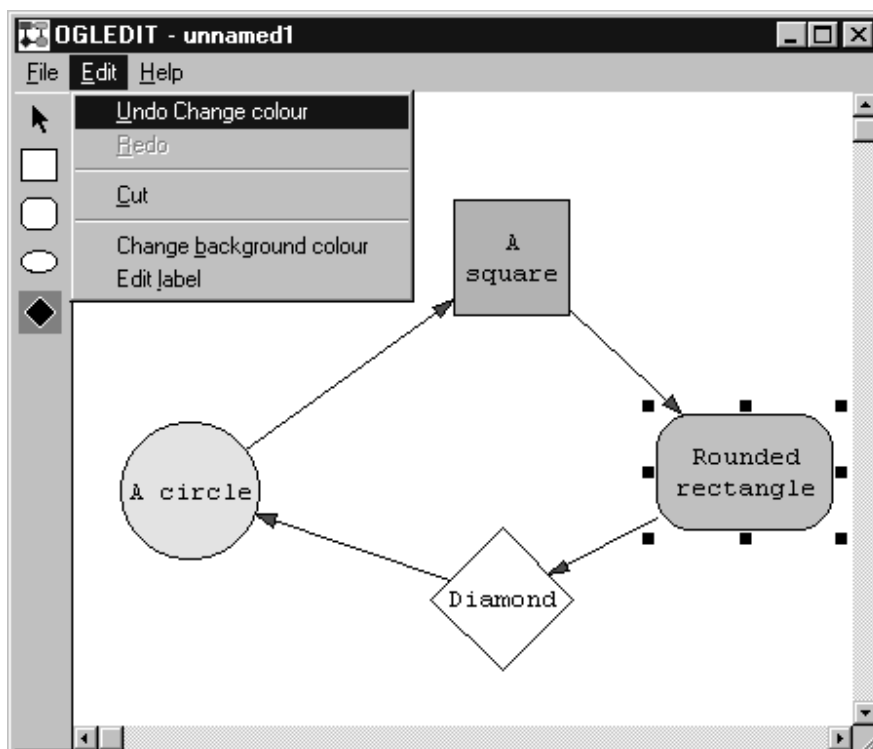
ogldiag.cpp wxDiagram implementation.

mfutils.h Metafile utilities header.

mfutils.cpp Metafile utilities implementation.

OGLEdit: a sample OGL application

OGLEdit is a sample OGL application that allows the user to draw, edit, save and load a few shapes. It should clarify aspects of OGL usage, and can act as a template for similar applications. OGLEdit can be found in `samples/ogledit` in the OGL distribution.



The wxWindows document/view model has been used in OGL, to reduce the amount of housekeeping logic required to get it up and running. OGLEdit also provides a demonstration of the Undo/Redo capability supported by the document/view classes, and how a typical application might implement this feature.

OGLEdit files

OGLEdit comprises the following source files.

- `doc.h`, `doc.cpp`: `MyDiagram`, `DiagramDocument`, `DiagramCommand`, `MyEvtHandler` classes related to diagram functionality and documents.
- `view.h`, `view.cpp`: `MyCanvas`, `DiagramView` classes related to visualisation of the diagram.
- `ogledit.h`, `ogledit.cpp`: `MyFrame`, `MyApp` classes related to the overall

application.

- palette.h, palette.cpp: EditorToolPalette implementing the shape palette.

How OGLEdit works

OGLEdit defines a DiagramDocument class, each of instance of which holds a MyDiagram member which itself contains the shapes.

In order to implement specific mouse behaviour for shapes, a class MyEvtHandler is defined which is 'plugged into' each shape when it is created, instead of overriding each shape class individually. This event handler class also holds a label string.

The DiagramCommand class is the key to implementing Undo/Redo. Each instance of DiagramCommand stores enough information about an operation (create, delete, change colour etc.) to allow it to carry out (or undo) its command.

Apart from menu commands, another way commands are initiated is by the user left-clicking on the canvas or right-dragging on a node. MyCanvas::OnLeftClick in view.cpp shows how the appropriate wxClassInfo is passed to a DiagramCommand, to allow DiagramCommand::Do to create a new shape given the wxClassInfo.

The MyEvtHandler right-drag methods in doc.cpp implement drawing a line between two shapes, detecting where the right mouse button was released and looking for a second shape. Again, a new DiagramCommand instance is created and passed to the command processor to carry out the command.

DiagramCommand::Do and DiagramCommand::Undo embody much of the interesting interaction with the OGL library. A complication of note when implementing undo is the problem of deleting a node shape which has one or more arcs attached to it. If you delete the node, the arc(s) should be deleted too. But multiple arc deletion represents more information that can be incorporated in the existing DiagramCommand scheme. OGLEdit copes with this by treating each arc deletion as a separate command, and sending Cut commands recursively, providing an undo path. Undoing such a Cut will only undo one command at a time - not a one to one correspondence with the original command - but it's a reasonable compromise and preserves Do/Undo while keeping our DiagramCommand class simple.

Possible enhancements

OGLEdit is very simplistic and does not employ the more advanced features of OGL, such as:

- attachment points (arcs are drawn to particular points on a shape)
- metafile and bitmaps shapes
- divided rectangles
- composite shapes, and constraints

- creating labels in shape regions
- arc labels (OGL has support for three movable labels per arc)
- spline and multiple-segment line arcs
- adding annotations to node and arc shapes
- line-straightening (supported by OGL) and alignment (not supported directly by OGL)

These could be added to OGLEdit, at the risk of making it a less useful example for beginners.

Class reference

These are the main OGL classes.

wxOGLConstraint

wxCompositeShape overview (p. 65)

An `wxOGLConstraint` object helps specify how child shapes are laid out with respect to siblings and parents.

Derived from

`wxObject`

See also

wxCompositeShape (p. 19)

wxOGLConstraint::wxOGLConstraint

wxOGLConstraint()

Default constructor.

wxOGLConstraint(int type, wxShape *constraining, wxList& constrained)

Constructor.

Parameters

constraining

The shape which is used as the reference for positioning the *constrained* objects.

constrained

Contains a list of `wxShapes` which are to be constrained (with respect to *constraining*) using *type*.

type

Can be one of:

- **gyCONSTRAINT_CENTRED_VERTICALLY**: the Y co-ordinates of the centres of the bounding boxes of the constrained objects and the constraining object will be the same

- **gyCONSTRAINT_CENTRED_HORIZONTALLY**: the X co-ordinates of the centres of the bounding boxes of the constrained objects and the constraining object will be the same
- **gyCONSTRAINT_CENTRED_BOTH**: the co-ordinates of the centres of the bounding boxes of the constrained objects and the constraining object will be the same
- **gyCONSTRAINT_LEFT_OF**: the X co-ordinates of the right hand vertical edges of the bounding boxes of the constrained objects will be less than the X co-ordinate of the left hand vertical edge of the bounding box of the constraining object
- **gyCONSTRAINT_RIGHT_OF**: the X co-ordinates of the left hand vertical edges of the bounding boxes of the constrained objects will be greater than the X co-ordinate of the right hand vertical edge of the bounding box of the constraining object
- **gyCONSTRAINT_ABOVE**: the Y co-ordinates of the bottom horizontal edges of the bounding boxes of the constrained objects will be less than the Y co-ordinate of the top horizontal edge of the bounding box of the constraining object
- **gyCONSTRAINT_BELOW**: the Y co-ordinates of the top horizontal edges of the bounding boxes of the constrained objects will be greater than the X co-ordinate of the bottom horizontal edge of the bounding box of the constraining object
- **gyCONSTRAINT_ALIGNED_TOP**: the Y co-ordinates of the top horizontal edges of the bounding boxes of the constrained objects will be the same as the Y co-ordinate of the top horizontal edge of the bounding box of the constraining object
- **gyCONSTRAINT_ALIGNED_BOTTOM**: the Y co-ordinates of the bottom horizontal edges of the bounding boxes of the constrained objects will be the same as the Y co-ordinate of the bottom horizontal edge of the bounding box of the constraining object
- **gyCONSTRAINT_ALIGNED_LEFT**: the X co-ordinates of the left hand vertical edges of the bounding boxes of the constrained objects will be the same as the X co-ordinate of the left hand vertical edge of the bounding box of the constraining object
- **gyCONSTRAINT_ALIGNED_RIGHT**: the X co-ordinates of the right hand vertical edges of the bounding boxes of the constrained objects will be the same as the X co-ordinate of the right hand vertical edge of the bounding box of the constraining object
- **gyCONSTRAINT_MIDALIGNED_TOP**: the Y co-ordinates of the centres of the bounding boxes of the constrained objects will be the same as the Y co-ordinate of the top horizontal edge of the bounding box of the constraining object

- **gyCONSTRAINT_MIDALIGNED_BOTTOM**: the Y co-ordinates of the centres of the bounding boxes of the constrained objects will be the same as the Y co-ordinate of the bottom horizontal edge of the bounding box of the constraining object
- **gyCONSTRAINT_MIDALIGNED_LEFT**: the X co-ordinates of the centres of the bounding boxes of the constrained objects will be the same as the X co-ordinate of the left hand vertical edge of the bounding box of the constraining object
- **gyCONSTRAINT_MIDALIGNED_RIGHT**: the X co-ordinates of the centres of the bounding boxes of the constrained objects will be the same as the X co-ordinate of the right hand vertical edge of the bounding box of the constraining object

wxOGLConstraint::~~wxOGLConstraint**~wxOGLConstraint()**

Destructor.

wxOGLConstraint::Equals**bool Equals(double x, double y)**

Returns TRUE if x and y are approximately equal (for the purposes of evaluating the constraint).

wxOGLConstraint::Evaluate**bool Evaluate()**

Evaluates this constraint, returning TRUE if anything changed.

wxOGLConstraint::SetSpacing**void SetSpacing(double x, double y)**

Sets the horizontal and vertical spacing for the constraint.

wxBitmapShape

Draws a bitmap (non-resizable).

Derived from*wxRectangleShape* (p. 34)

wxBitmapShape::wxBitmapShape**wxBitmapShape()**

Constructor.

wxBitmapShape::~~wxBitmapShape**~wxBitmapShape()**

Destructor.

wxBitmapShape::GetBitmap**wxBitmap& GetBitmap() const**

Returns a reference to the bitmap associated with this shape.

wxBitmapShape::GetFilename**wxString GetFilename() const**

Returns the bitmap filename.

wxBitmapShape::SetBitmap**void SetBitmap(const wxBitmap& *bitmap*)**

Sets the bitmap associated with this shape. You can delete the bitmap from the calling application, since reference counting will take care of holding on to the internal bitmap data.

wxBitmapShape::SetFilename**void SetFilename(const wxString& *filename*)**

Sets the bitmap filename.

wxDiagram

Encapsulates an entire diagram, with methods for reading/writing and drawing. A diagram has an associated wxShapeCanvas.

Derived from

wxObject

See also*wxShapeCanvas* (p. 52)

wxDiagram::wxDiagram**wxDiagram()**

Constructor.

wxDiagram::~~wxDiagram**~wxDiagram()**

Destructor.

wxDiagram::AddShape**void AddShape(wxShape* shape, wxShape *addAfter = NULL)**

Adds a shape to the diagram. If *addAfter* is non-NULL, the shape will be added after this one.

wxDiagram::Clear**void Clear(wxDC& dc)**

Clears the specified device context.

wxDiagram::DeleteAllShapes**void DeletesAllShapes()**

Removes and deletes all shapes in the diagram.

wxDiagram::DrawOutline**void DrawOutline(wxDC& dc, double x1, double y1, double x2, double y2)**

Draws an outline rectangle on the current device context.

wxDiagram::FindShape**wxShape* FindShape(long id) const**

Returns the shape for the given identifier.

wxDiagram::GetCanvas**wxShapeCanvas* GetCanvas() const**

Returns the shape canvas associated with this diagram.

wxDiagram::GetCount**int GetCount() const**

Returns the number of shapes in the diagram.

wxDiagram::GetGridSpacing**double GetGridSpacing() const**

Returns the grid spacing.

wxDiagram::GetMouseTolerance**int GetMouseTolerance()**

Returns the tolerance within which a mouse move is ignored.

wxDiagram::GetShapeList**wxList* GetShapeList() const**

Returns a pointer to the internal shape list.

wxDiagram::GetQuickEditMode**bool GetQuickEditMode() const**

Returns quick edit mode.

wxDiagram::GetSnapToGrid**bool GetSnapToGrid() const**

Returns snap-to-grid mode.

wxDiagram::InsertShape**void InsertShape(wxShape *shape)**

Inserts a shape at the front of the shape list.

wxDiagram::LoadFile**bool LoadFile(const wxString& filename)**

Loads the diagram from a file.

wxDiagram::OnDatabaseLoad

void OnDatabaseLoad(wxExprDatabase& database)

Called just after the nodes and lines have been read from the wxExprDatabase. You may override this; the default member does nothing.

wxDiagram::OnDatabaseSave**void OnDatabaseSave(wxExprDatabase& database)**

Called just after the nodes and lines have been written to the wxExprDatabase. You may override this; the default member does nothing.

wxDiagram::OnHeaderLoad**bool OnHeaderLoad(wxExprDatabase& database, wxExpr& expr)**

Called to allow the 'diagram' header object to be read. The default member reads no further information. You may wish to override this to read version information, author name, etc.

wxDiagram::OnHeaderSave**bool OnHeaderSave(wxExprDatabase& database, wxExpr& expr)**

Called to allow instantiation of the 'diagram' header object. The default member writes no further information. You may wish to override this to include version information, author name, etc.

wxDiagram::OnShapeLoad**bool OnShapeLoad(wxExprDatabase& database, wxShape& shape, wxExpr& expr)**

Called to read the shape from the *expr*. You may override this, but call this function first. The default member calls ReadAttributes for the shape.

wxDiagram::OnShapeSave**bool OnShapeSave(wxExprDatabase& database, wxShape& shape, wxExpr& expr)**

Called to save the shape to the *expr* and *database*. You may override this, but call this function first. The default member calls WriteAttributes for the shape, appends the shape to the database, and if the shape is a composite, recursively calls OnShapeSave for its children.

wxDiagram::ReadContainerGeometry**void ReadContainerGeometry(wxExprDatabase& database)**

Reads container geometry from a wxExprDatabase, linking up nodes which are part of a composite. You probably won't need to redefine this.

wxDiagram::ReadLines**void ReadLines(wxExprDatabase& database)**

Reads lines from a wxExprDatabase. You probably won't need to redefine this.

wxDiagram::ReadNodes**void ReadNodes(wxExprDatabase& database)**

Reads nodes from a wxExprDatabase. You probably won't need to redefine this.

wxDiagram::RecentreAll**void RecentreAll(wxDC& dc)**

Make sure all text that should be centred, is centred.

wxDiagram::Redraw**void Redraw(wxDC& dc)**

Draws the shapes in the diagram on the specified device context.

wxDiagram::RemoveAllShapes**void RemoveAllShapes()**

Removes all shapes from the diagram but does not delete the shapes.

wxDiagram::RemoveShape**void RemoveShape(wxShape* shape)**

Removes the shape from the diagram (non-recursively) but does not delete it.

wxDiagram::SaveFile**bool SaveFile(const wxString& filename)**

Saves the diagram in a file.

wxDiagram::SetCanvas**void SetCanvas(wxShapeCanvas* canvas)**

Sets the canvas associated with this diagram.

wxDiagram::SetGridSpacing

void SetGridSpacing(double spacing)

Sets the grid spacing. The default is 5.

wxDiagram::SetMouseTolerance

void SetMouseTolerance(int tolerance)

Sets the tolerance within which a mouse move is ignored. The default is 3 pixels.

wxDiagram::SetQuickEditMode

void SetQuickEditMode(bool mode)

Sets quick-edit-mode on or off. In this mode, refreshes are minimized, but the diagram may need manual refreshing occasionally.

wxDiagram::SetSnapToGrid

void SetSnapToGrid(bool snap)

Sets snap-to-grid mode on or off. The default is on.

wxDiagram::ShowAll

void ShowAll(bool show)

Calls Show for each shape in the diagram.

wxDiagram::Snap

void Snap(double *x, double *y)

'Snaps' the coordinate to the nearest grid position, if snap-to-grid is on.

wxDrawnShape

Draws a pseudo-metafile shape, which can be loaded from a simple Windows metafile.

wxDrawnShape allows you to specify a different shape for each of four orientations (North, West, South and East). It also provides a set of drawing functions for programmatic drawing of a shape, so that during construction of the shape you can draw into it as if it were a device context.

Derived from

wxRectangleShape (p. 34)

See also *wxRectangleShape* (p. 34).

wxDrawnShape::wxDrawnShape**wxDrawnShape()**

Constructor.

wxDrawnShape::~~wxDrawnShape**~wxDrawnShape()**

Destructor.

wxDrawnShape::CalculateSize**void CalculateSize()**

Calculates the wxDrawnShape size from the current metafile. Call this after you have drawn into the shape.

wxDrawnShape::DestroyClippingRect**void DestroyClippingRect()**

Destroys the clipping rectangle. See also *wxDrawnShape::SetClippingRect* (p. 17).

wxDrawnShape::DrawArc**void DrawArc(const wxPoint& centrePoint, const wxPoint& startPoint, const wxPoint& endPoint)**

Draws an arc (see wxWindows documentation for details).

wxDrawnShape::DrawAtAngle**void DrawAtAngle(int angle)**

Sets the metafile for the given orientation, which can be one of:

- `ogIDRAWN_ANGLE_0`
- `ogIDRAWN_ANGLE_90`
- `ogIDRAWN_ANGLE_180`
- `ogIDRAWN_ANGLE_270`

See also *wxDrawnShape::GetAngle* (p. 17).

wxDrawnShape::DrawEllipticArc**void DrawEllipticArc(const wxRect& rect, double startAngle, double endAngle)**

Draws an elliptic arc (see wxWindows documentation for details).

wxDrawnShape::DrawLine

void DrawLine(const wxPoint& point1, const wxPoint& point2)

Draws a line from *point1* to *point2*.

wxDrawnShape::DrawLines

void DrawLines(int n, wxPoint& points[])

Draws *n* lines.

wxDrawnShape::DrawPoint

void DrawPoint(const wxPoint& point)

Draws a point.

wxDrawnShape::DrawPolygon

void DrawPolygon(int n, wxPoint& points[], int flags = 0)

Draws a polygon. *flags* can be one or more of **ogIMETAFLAGS_OUTLINE** (use this polygon for the drag outline) and **ogIMETAFLAGS_ATTACHMENTS** (use the vertices of this polygon for attachments).

wxDrawnShape::DrawRectangle

void DrawRectangle(const wxRect& rect)

Draws a rectangle.

wxDrawnShape::DrawRoundedRectangle

void DrawRoundedRectangle(const wxRect& rect, double radius)

Draws a rounded rectangle. *radius* is the corner radius. If *radius* is negative, it expresses the radius as a proportion of the smallest dimension of the rectangle.

wxDrawnShape::DrawSpline

void DrawSpline(int n, wxPoint& points[])

Draws a spline curve.

wxDrawnShape::DrawText

void DrawText(const wxString& text, const wxPoint& point)

Draws text at the given point.

wxDrawnShape::GetAngle

int GetAngle() const

Returns the current orientation, which can be one of:

- `ogIDRAWN_ANGLE_0`
- `ogIDRAWN_ANGLE_90`
- `ogIDRAWN_ANGLE_180`
- `ogIDRAWN_ANGLE_270`

See also *wxDrawnShape::DrawAtAngle* (p. 15).

wxDrawnShape::GetMetaFile

wxPseudoMetaFile& GetMetaFile() const

Returns a reference to the internal 'pseudo-metafile'.

wxDrawnShape::GetRotation

double GetRotation() const

Returns the current rotation of the shape in radians.

wxDrawnShape::LoadFromMetaFile

bool LoadFromMetaFile(const wxString& filename)

Loads a (very simple) Windows metafile, created for example by Top Draw, the Windows shareware graphics package.

wxDrawnShape::Rotate

void Rotate(double x, double y, double theta)

Rotate about the given axis by the given amount in radians.

wxDrawnShape::SetClippingRect

void SetClippingRect(const wxRect& rect)

Sets the clipping rectangle. See also *wxDrawnShape::DestroyClippingRect* (p. 15).

wxDrawnShape::SetDrawnBackgroundColour

void SetDrawnBackgroundColour(const wxColour& colour)

Sets the current background colour for the current metafile.

wxDrawnShape::SetDrawnBackgroundMode

void SetDrawnBackgroundMode(int mode)

Sets the current background mode for the current metafile.

wxDrawnShape::SetDrawnBrush

void SetDrawnBrush(wxPen* pen, bool isOutline = FALSE)

Sets the pen for this metafile. If *isOutline* is TRUE, this pen is taken to indicate the outline (and if the outline pen is changed for the whole shape, the pen will be replaced with the outline pen).

wxDrawnShape::SetDrawnFont

void SetDrawnFont(wxFont* font)

Sets the current font for the current metafile.

wxDrawnShape::SetDrawnPen

void SetDrawnPen(wxPen* pen, bool isOutline = FALSE)

Sets the pen for this metafile. If *isOutline* is TRUE, this pen is taken to indicate the outline (and if the outline pen is changed for the whole shape, the pen will be replaced with the outline pen).

wxDrawnShape::SetDrawnTextColour

void SetDrawnTextColour(const wxColour& colour)

Sets the current text colour for the current metafile.

wxDrawnShape::Scale

void Scale(double sx, double sy)

Scales the shape by the given amount.

wxDrawnShape::SetSaveToFile

void SetSaveToFile(bool save)

If *save* is TRUE, the image will be saved along with the shape's other attributes. The reason why this might not be desirable is that if there are many shapes with the same

image, it would be more efficient for the application to save one copy, and not duplicate the information for every shape. The default is TRUE.

wxDrawnShape::Translate

void Translate(double x, double y)

Translates the shape by the given amount.

wxCircleShape

An wxEllipseShape whose width and height are the same.

Derived from

wxEllipseShape (p. 26).

wxCircleShape::wxCircleShape

wxCircleShape(double width = 0.0)

Constructor.

wxCircleShape::~~wxCircleShape

~wxCircleShape()

Destructor.

wxCompositeShape

This is an object with a list of child objects, and a list of size and positioning constraints between the children.

Derived from

wxRectangleShape (p. 34)

See also

wxCompositeShape overview (p. 65)

wxCompositeShape::wxCompositeShape

wxCompositeShape()

Constructor.

wxCompositeShape::~~wxCompositeShape**~wxCompositeShape()**

Destructor.

wxCompositeShape::AddChild**void AddChild(wxShape *child, wxShape *addAfter = NULL)**

Adds a child shape to the composite. If *addAfter* is non-NULL, the shape will be added after this shape.

wxCompositeShape::AddConstraint**wxOGLConstraint * AddConstraint(wxOGLConstraint *constraint)****wxOGLConstraint * AddConstraint(int type, wxShape *constraining, wxList&constrained)****wxOGLConstraint * AddConstraint(int type, wxShape *constraining, wxShape *constrained)**

Adds a constraint to the composite.

wxCompositeShape::CalculateSize**void CalculateSize()**

Calculates the size and position of the composite based on child sizes and positions.

wxCompositeShape::ContainsDivision**bool FindContainerImage(wxDivisionShape *division)**

Returns TRUE if *division* is a descendant of this container.

wxCompositeShape::DeleteConstraint**void DeleteConstraint(wxOGLConstraint *constraint)**

Deletes constraint from composite.

wxCompositeShape::DeleteConstraintsInvolvingChild**void DeleteConstraintsInvolvingChild(wxShape *child)**

This function deletes constraints which mention the given child. Used when deleting a child from the composite.

wxCompositeShape::FindConstraint**wxOGLConstraint * FindConstraint(long id, wxCompositeShape **actualComposite)**

Finds the constraint with the given id, also returning the actual composite the constraint was in, in case that composite was a descendant of this composite.

wxCompositeShape::FindContainerImage**wxShape * FindContainerImage()**

Finds the image used to visualize a container. This is any child of the composite that is not in the divisions list.

wxCompositeShape::GetConstraints**wxList& GetConstraints() const**

Returns a reference to the list of constraints.

wxCompositeShape::GetDivisions**wxList& GetDivisions() const**

Returns a reference to the list of divisions.

wxCompositeShape::MakeContainer**void MakeContainer()**

Makes this composite into a container by creating one child wxDivisionShape.

wxCompositeShape::OnCreateDivision**wxDivisionShape * OnCreateDivision()**

Called when a new division shape is required. Can be overridden to allow an application to use a different class of division.

wxCompositeShape::Recompute**bool Recompute()**

Recomputes any constraints associated with the object. If FALSE is returned, the constraints could not be satisfied (there was an inconsistency).

wxCompositeShape::RemoveChild**void RemoveChild(wxShape *child)**

Removes the child from the composite and any constraint relationships, but does not delete the child.

wxDividedShape

A wxDividedShape is a rectangle with a number of vertical divisions. Each division may have its text formatted with independent characteristics, and the size of each division relative to the whole image may be specified.

Derived from

wxRectangleShape (p. 34)

See also

wxDividedShape overview (p. 63)

wxDividedShape::wxDividedShape

wxDividedShape(double *width* = 0.0, double *height* = 0.0)

Constructor.

wxDividedShape::~~wxDividedShape

~wxDividedShape()

Destructor.

wxDividedShape::EditRegions

void EditRegions()

Edit the region colours and styles.

wxDividedShape::SetRegionSizes

void SetRegionSizes()

Set all region sizes according to proportions and this object total size.

wxDivisionShape

A division shape is like a composite in that it can contain further objects, but is used exclusively to divide another shape into regions, or divisions. A wxDivisionShape is never free-standing.

Derived from

wxCompositeShape (p. 19)

See also

wxCompositeShape overview (p. 65)

wxDivisionShape::wxDivisionShape

wxDivisionShape()

Constructor.

wxDivisionShape::~~wxDivisionShape

~wxDivisionShape()

Destructor.

wxDivisionShape::AdjustBottom

void AdjustBottom(double bottom, bool test)

Adjust a side, returning FALSE if it's not physically possible to adjust it to this point.

wxDivisionShape::AdjustLeft

void AdjustLeft(double left, bool test)

Adjust a side, returning FALSE if it's not physically possible to adjust it to this point.

wxDivisionShape::AdjustRight

void AdjustRight(double right, bool test)

Adjust a side, returning FALSE if it's not physically possible to adjust it to this point.

wxDivisionShape::AdjustTop

void AdjustTop(double top, bool test)

Adjust a side, returning FALSE if it's not physically possible to adjust it to this point.

wxDivisionShape::Divide

void Divide(int direction)

Divide this division into two further divisions, horizontally (*direction* is wxHORIZONTAL) or vertically (*direction* is wxVERTICAL).

wxDivisionShape::EditEdge**void EditEdge(int side)**

Interactively edit style of left or top side.

wxDivisionShape::GetBottomSide**wxDivisionShape * GetBottomSide()**

Returns a pointer to the division on the bottom side of this division.

wxDivisionShape::GetHandleSide**int GetHandleSide()**

Returns the side which the handle appears on (DIVISION_SIDE_LEFT or DIVISION_SIDE_TOP).

wxDivisionShape::GetLeftSide**wxDivisionShape * GetLeftSide()**

Returns a pointer to the division on the left side of this division.

wxDivisionShape::GetLeftSideColour**wxString GetLeftSideColour()**

Returns a pointer to the colour used for drawing the left side of the division.

wxDivisionShape::GetLeftSidePen**wxPen * GetLeftSidePen()**

Returns a pointer to the pen used for drawing the left side of the division.

wxDivisionShape::GetRightSide**wxDivisionShape * GetRightSide()**

Returns a pointer to the division on the right side of this division.

wxDivisionShape::GetTopSide**wxDivisionShape * GetTopSide()**

Returns a pointer to the division on the top side of this division.

wxDivisionShape::GetTopSideColour

wxString GetTopSideColour()

Returns a pointer to the colour used for drawing the top side of the division.

wxDivisionShape::GetTopSidePen**wxPen * GetTopSidePen()**

Returns a pointer to the pen used for drawing the left side of the division.

wxDivisionShape::ResizeAdjoining**void ResizeAdjoining(int side, double newPos, bool test)**

Resize adjoining divisions at the given side. If *test* is TRUE, just see whether it's possible for each adjoining region, returning FALSE if it's not.

side can be one of:

- DIVISION_SIDE_NONE
- DIVISION_SIDE_LEFT
- DIVISION_SIDE_TOP
- DIVISION_SIDE_RIGHT
- DIVISION_SIDE_BOTTOM

wxDivisionShape::PopupMenu**void PopupMenu(double x, double y)**

Popup the division menu.

wxDivisionShape::SetBottomSide**void SetBottomSide(wxDivisionShape *shape)**

Set the pointer to the division on the bottom side of this division.

wxDivisionShape::SetHandleSide**int SetHandleSide()**

Sets the side which the handle appears on (DIVISION_SIDE_LEFT or DIVISION_SIDE_TOP).

wxDivisionShape::SetLeftSide**void SetLeftSide(wxDivisionShape *shape)**

Set the pointer to the division on the left side of this division.

wxDivisionShape::SetLeftSideColour**void SetLeftSideColour(const wxString& colour)**

Sets the colour for drawing the left side of the division.

wxDivisionShape::SetLeftSidePen**void SetLeftSidePen(wxPen *pen)**

Sets the pen for drawing the left side of the division.

wxDivisionShape::SetRightSide**void SetRightSide(wxDivisionShape *shape)**

Set the pointer to the division on the right side of this division.

wxDivisionShape::SetTopSide**void SetTopSide(wxDivisionShape *shape)**

Set the pointer to the division on the top side of this division.

wxDivisionShape::SetTopSideColour**void SetTopSideColour(const wxString& colour)**

Sets the colour for drawing the top side of the division.

wxDivisionShape::SetTopSidePen**void SetTopSidePen(wxPen *pen)**

Sets the pen for drawing the top side of the division.

wxEllipseShape

The wxEllipseShape behaves similarly to the wxRectangleShape but is elliptical.

Derived from

wxShape (p. 35)

wxEllipseShape::wxEllipseShape

wxEllipseShape(double *width* = 0.0, double *height* = 0.0)

Constructor.

wxEllipseShape::~~wxEllipseShape

~wxEllipseShape()

Destructor.

wxLineShape

A wxLineShape may be attached to two nodes; it may be segmented, in which case a control point is drawn for each joint.

A wxLineShape may have arrows at the beginning, end and centre.

Derived from

wxShape (p. 35)

wxLineShape::wxLineShape

wxLineShape()

Constructor.

Usually you will call *wxLineShape::MakeLineControlPoints* (p. 31) to specify the number of segments in the line.

wxLineShape::~~wxLineShape

~wxLineShape()

Destructor.

wxLineShape::AddArrow

void AddArrow(WXTYPE *type*, bool *end* = ARROW_POSITION_END, double *arrowSize* = 10.0, double *xOffset* = 0.0, const wxString& *name* = "", wxPseudoMetaFile **mf* = NULL, long *arrowId* = -1)

Adds an arrow (or annotation) to the line.

type may currently be one of:

ARROW_HOLLOW_CIRCLE Hollow circle.

ARROW_FILLED_CIRCLE Filled circle.

ARROW_ARROW Conventional arrowhead.

ARROW_SINGLE_OBLIQUE Single oblique stroke.

ARROW_DOUBLE_OBLIQUE Double oblique stroke.

ARROW_DOUBLE_METAFILE Custom arrowhead.

end may currently be one of:

ARROW_POSITION_END Arrow appears at the end.

ARROW_POSITION_START Arrow appears at the start.

arrowSize specifies the length of the arrow.

xOffset specifies the offset from the end of the line.

name specifies a name for the arrow.

mf can be a `wxPseudoMetaFile`, perhaps loaded from a simple Windows metafile.

arrowId is the id for the arrow.

wxLineShape::AddArrowOrdered

void AddArrowOrdered(wxArrowHead *arrow, wxList& referenceList, int end)

Add an arrowhead in the position indicated by the reference list of arrowheads, which contains all legal arrowheads for this line, in the correct order. E.g.

```
Reference list:      a b c d e
Current line list:  a d
```

Add c, then line list is: a c d.

If no legal arrowhead position, return `FALSE`. Assume reference list is for one end only, since it potentially defines the ordering for any one of the 3 positions. So we don't check the reference list for arrowhead position.

wxLineShape::ClearArrow

bool ClearArrow(const wxString& name)

Delete the arrow with the given name.

wxLineShape::ClearArrowsAtPosition

void ClearArrowsAtPosition(int position = -1)

Delete the arrows at the specified position, or at any position if *position* is -1.

wxLineShape::DrawArrow

void DrawArrow(ArrowHead *arrow, double xOffset, bool proportionalOffset)

Draws the given arrowhead (or annotation).

wxLineShape::DeleteArrowHead

bool DeleteArrowHead(long arrowId)

bool DeleteArrowHead(int position, const wxString& name)

Delete arrowhead by id or position and name.

wxLineShape::DeleteLineControlPoint

bool DeleteLineControlPoint()

Deletes an arbitrary point on the line.

wxLineShape::DrawArrows

void DrawArrows(wxDC& dc)

Draws all arrows.

wxLineShape::DrawRegion

void DrawRegion(wxDC& dc, wxShapeRegion *region, double x, double y)

Format one region at this position.

wxLineShape::EraseRegion

void EraseRegion(wxDC& dc, wxShapeRegion *region, double x, double y)

Format one region at this position.

wxLineShape::FindArrowHead

wxArrowHead * FindArrowHead(long arrowId)

wxArrowHead * FindArrowHead(int position, const wxString& name)

Find arrowhead by id or position and name.

wxLineShape::FindLineEndpoints

void FindLineEndpoints(double *fromX, double *fromY, double *toX, double *toY)

Finds the x, y points at the two ends of the line. This function can be used by e.g. line-routing routines to get the actual points on the two node images where the lines will be drawn to/from.

wxLineShape::FindLinePosition**int FindLinePosition(double x, double y)**

Find which position we're talking about at this x, y. Returns ARROW_POSITION_START, ARROW_POSITION_MIDDLE, ARROW_POSITION_END.

wxLineShape::FindMinimumWidth**double FindMinimumWidth()**

Finds the horizontal width for drawing a line with arrows in minimum space. Assume arrows at end only.

wxLineShape::FindNth**void FindNth(wxShape *image, int *nth, int *noArcs, bool incoming)**

Finds the position of the line on the given object. Specify whether incoming or outgoing lines are being considered with *incoming*.

wxLineShape::GetAttachmentFrom**int GetAttachmentFrom() const**

Returns the attachment point on the 'from' node.

wxLineShape::GetAttachmentTo**int GetAttachmentTo() const**

Returns the attachment point on the 'to' node.

wxLineShape::GetEnds**void GetEnds(double *x1, double *y1, double *x2, double *y2)**

Gets the visible endpoints of the lines for drawing between two objects.

wxLineShape::GetFrom**wxShape * GetFrom() const**

Gets the 'from' object.

wxLineShape::GetLabelPosition**void GetLabelPosition(int position, double *x, double *y)**

Get the reference point for a label. Region *x* and *y* are offsets from this. position is 0 (middle), 1 (start), 2 (end).

wxLineShape::GetNextControlPoint

wxPoint * GetNextControlPoint(wxShape *shape)

Find the next control point in the line after the start/end point, depending on whether the shape is at the start or end.

wxLineShape::GetTo

wxShape * GetTo()

Gets the 'to' object.

wxLineShape::Initialise

void Initialise()

Initialises the line object.

wxLineShape::InsertLineControlPoint

void InsertLineControlPoint()

Inserts a control point at an arbitrary position.

wxLineShape::IsEnd

bool IsEnd(wxShape *shape)

Returns TRUE if *shape* is at the end of the line.

wxLineShape::IsSpline

bool IsSpline()

Returns TRUE if a spline is drawn through the control points, and FALSE otherwise.

wxLineShape::MakeLineControlPoints

void MakeLineControlPoints(int n)

Make a given number of control points (minimum of two).

wxLineShape::OnMoveLink

void OnMoveLink(wxDC& dc, bool moveControlPoints = TRUE)

Called when a connected object has moved, to move the link to correct position.

wxLineShape::SetAttachmentFrom**void SetAttachmentTo(int *fromAttach*)**

Sets the 'from' shape attachment.

wxLineShape::SetAttachments**void SetAttachments(int *fromAttach*, int *toAttach*)**

Specifies which object attachment points should be used at each end of the line.

wxLineShape::SetAttachmentTo**void SetAttachmentTo(int *toAttach*)**

Sets the 'to' shape attachment.

wxLineShape::SetEnds**void SetEnds(double *x1*, double *y1*, double *x2*, double *y2*)**

Sets the end positions of the line.

wxLineShape::SetFrom**void SetFrom(wxShape **object*)**

Sets the 'from' object for the line.

wxLineShape::SetIgnoreOffsets**void SetIgnoreOffsets(bool *ignore*)**

Tells the shape whether to ignore offsets from the end of the line when drawing.

wxLineShape::SetSpline**void SetSpline(bool *spline*)**

Specifies whether a spline is to be drawn through the control points (TRUE), or a line (FALSE).

wxLineShape::SetTo**void SetTo(wxShape **object*)**

Sets the 'to' object for the line.

wxLineShape::Straighten**void Straighten**(wxDC* *dc* = *NULL*)

Straighten verticals and horizontals. *dc* is optional.

wxLineShape::Unlink**void Unlink**()

Unlinks the line from the nodes at either end.

wxPolygonShape

A wxPolygonShape's shape is defined by a number of points passed to the object's constructor. It can be used to create new shapes such as diamonds and triangles.

Derived from

wxShape (p. 35)

wxPolygonShape::wxPolygonShape**wxPolygonShape**(void)

Constructor. Call *wxPolygonShape::Create* (p. 33) to specify the polygon's vertices.

wxPolygonShape::~~wxPolygonShape**~wxPolygonShape**()

Destructor.

wxPolygonShape::Create**void Create**(wxList* *points*)

Takes a list of wxRealPoints; each point is an *offset* from the centre. The polygon's destructor will delete these points, so do not delete them yourself.

wxPolygonShape::AddPolygonPoint**void AddPolygonPoint**(int *pos* = 0)

Add a control point after the given point.

wxPolygonShape::CalculatePolygonCentre

void CalculatePolygonCentre()

Recalculates the centre of the polygon.

wxPolygonShape::DeletePolygonPoint**void DeletePolygonPoint(int pos = 0)**

Deletes a control point.

wxPolygonShape::GetPoints**wxList * GetPoints()**

Returns a pointer to the internal list of polygon vertices (wxRealPoints).

wxPolygonShape::UpdateOriginalPoints**void UpdateOriginalPoints()**

If we've changed the shape, must make the original points match the working points with this function.

wxRectangleShape

The wxRectangleShape has rounded or square corners.

Derived from

wxShape (p. 35)

wxRectangleShape::wxRectangleShape**wxRectangleShape(double width = 0.0, double height = 0.0)**

Constructor.

wxRectangleShape::~~wxRectangleShape**~wxRectangleShape()**

Destructor.

wxRectangleShape::SetCornerRadius**void SetCornerRadius(double radius)**

Sets the radius of the rectangle's rounded corners. If the radius is zero, a non-rounded

rectangle will be drawn. If the radius is negative, the value is the proportion of the smaller dimension of the rectangle.

wxPseudoMetaFile

A simple metafile-like class which can load data from a Windows metafile on all platforms.

Derived from

wxObject

wxShape

The wxShape is the top-level, abstract object that all other objects are derived from. All common functionality is represented by wxShape's members, and overridden members that appear in derived classes and have behaviour as documented for wxShape, are not documented separately.

Derived from

wxShapeEvtHandler (p. 57)

wxShape::wxShape

wxShape(wxShapeCanvas* canvas = NULL)

Constructs a new wxShape.

wxShape::~~wxShape

~wxShape()

Destructor.

wxShape::AddLine

void AddLine(wxLineShape* line, wxShape* other, int attachFrom = 0, int attachTo = 0, int positionFrom = -1, int positionTo = -1)

Adds a line between the specified canvas shapes, at the specified attachment points.

The position in the list of lines at each end can also be specified, so that the line will be drawn at a particular point on its attachment point.

wxShape::AddRegion

void AddRegion(wxShapeRegion* region)

Adds a region to the shape.

wxShape::AddText

void AddText(const wxString& *string*)

Adds a line of text to the shape's default text region.

wxShape::AddToCanvas

void AddToCanvas(wxShapeCanvas* *theCanvas*, wxShape* *addAfter*=NULL)

Adds the shape to the canvas's shape list. If *addAfter* is non-NULL, will add the shape after this one.

wxShape::AncestorSelected

bool AncestorSelected() const

TRUE if the shape's ancestor is currently selected.

wxShape::ApplyAttachmentOrdering

void ApplyAttachmentOrdering(wxList& *linesToSort*)

Applies the line ordering in *linesToSort* to the shape, to reorder the way lines are attached.

wxShape::AssignNewIds

void AssignNewIds()

Assigns new ids to this image and its children.

wxShape::Attach

void Attach(wxShapeCanvas* *can*)

Sets the shape's internal canvas pointer to point to the given canvas.

wxShape::AttachmentIsValid

bool AttachmentIsValid(int *attachment*) const

Returns TRUE if *attachment* is a valid attachment point.

wxShape::AttachmentSortTest

bool AttachmentSortTest(int *attachment*, const wxRealPoint& *pt1*, const wxRealPoint& *pt2*) const

Returns TRUE if *pt1* is less than or equal to *pt2*, in the sense that one point comes before another on an edge of the shape. *attachment* is the attachment point (side) in question.

This function is used in *wxShape::MoveLineToNewAttachment* (p. 46) to determine the new line ordering.

wxShape::CalcSimpleAttachment

wxRealPoint CalcSimpleAttachment(const wxRealPoint& *pt1*, const wxRealPoint& *pt2*, int *nth*, int *noArcs*, wxLineShape* *line*)

Assuming the attachment lies along a vertical or horizontal line, calculates the position on that point.

Parameters

pt1

The first point of the line representing the edge of the shape.

pt2

The second point of the line representing the edge of the shape.

nth

The position on the edge (for example there may be 6 lines at this attachment point, and this may be the 2nd line).

noArcs

The number of lines at this edge.

line

The line shape.

Remarks

This function expects the line to be either vertical or horizontal, and determines which.

wxShape::CalculateSize

void CalculateSize()

Called to calculate the shape's size if dependent on children sizes.

wxShape::ClearAttachments

void ClearAttachments()

Clears internal custom attachment point shapes (of class *wxAttachmentPoint*).

wxShape::ClearRegions**void ClearRegions()**

Clears the wxShapeRegions from the shape.

wxShape::ClearText**void ClearText(int regionId = 0)**

Clears the text from the specified text region.

wxShape::Constrain**bool Constrain()**

Calculates the shape's constraints (if any). Applicable only to wxCompositeShape, does nothing if the shape is of a different class.

wxShape::Copy**void Copy(wxShape& copy)**

Copy this shape into *copy*. Every derived class must have one of these, and each Copy implementation must call the derived class's implementation to ensure everything is copied. See also *wxShape::CreateNewCopy* (p. 38).

wxShape::CreateNewCopy**wxShape* CreateNewCopy(bool resetMapping = TRUE, bool recompute = TRUE)**

Creates and returns a new copy of this shape (calling *wxShape::Copy* (p. 38)). Do not override this function.

This function should always be used to create a new copy, since it must do special processing for copying constraints associated with constraints.

If *resetMapping* is TRUE, a mapping table used for complex shapes is reset; this may not be desirable if the shape being copied is a child of a composite (and so the mapping table is in use).

If *recompute* is TRUE, *wxShape::Recompute* (p. 47) is called for the new shape.

Remarks

This function uses the wxWindows dynamic object creation system to create a new shape of the same type as 'this', before calling Copy.

If the event handler for this shape is not the same as the shape itself, the event handler is also copied using *wxShapeEvtHandler::CreateNewCopy* (p. 58).

wxShape::DeleteControlPoints**void DeleteControlPoints()**

Deletes the control points (or handles) for the shape. Does not redraw the shape.

wxShape::Detach**void Detach()**

Disassociates the shape from its canvas by setting the internal shape canvas pointer to NULL.

wxShape::Draggable**bool Draggable()**

TRUE if the shape may be dragged by the user.

wxShape::Draw**void Draw(wxDC& dc)**

Draws the whole shape and any lines attached to it.

Do not override this function: override OnDraw, which is called by this function.

wxShape::DrawContents**void DrawContents(wxDC& dc)**

Draws the internal graphic of the shape (such as text).

Do not override this function: override OnDrawContents, which is called by this function.

wxShape::DrawLinks**void DrawLinks(wxDC& dc, int attachment = -1)**

Draws any lines linked to this shape.

wxShape::Erase**void Erase(wxDC& dc)**

Erases the shape, but does not repair damage caused to other shapes.

wxShape::EraseContents**void EraseContents(wxDC& dc)**

Erases the shape contents, that is, the area within the shape's minimum bounding box.

wxShape::EraseLinks

void EraseLinks(wxDC& dc, int attachment = -1)

Erases links attached to this shape, but does not repair damage caused to other shapes.

wxShape::FindRegion

wxShape * FindRegion(const wxString& regionName, int *regionId)

Finds the actual image ('this' if non-composite) and region id for the given region name.

wxShape::FindRegionNames

void FindRegionNames(wxStringList& list)

Finds all region names for this image (composite or simple). Supply an empty string list.

wxShape::Flash

void Flash()

Flashes the shape.

wxShape::FormatText

void FormatText(const wxString& s, int i = 0)

Reformats the given text region; defaults to formatting the default region.

wxShape::GetAttachmentMode

bool GetAttachmentMode() const

Returns the attachment mode, which is TRUE if attachments are used, FALSE otherwise (in which case lines will be drawn as if to the centre of the shape). See *wxShape::SetAttachmentMode* (p. 48).

wxShape::GetAttachmentPosition

bool GetAttachmentPosition(int attachment, double* x, double* y, int nth = 0, int noArcs = 1, wxLineShape* line = NULL)

Gets the position at which the given attachment point should be drawn.

If *attachment* isn't found among the attachment points of the shape, returns FALSE.

wxShape::GetBoundingBoxMax

void GetBoundingBoxMax(double *width, double *height)

Gets the maximum bounding box for the shape, taking into account external features such as shadows.

wxShape::GetBoundingBoxMin

void GetBoundingBoxMin(double *width, double *height)

Gets the minimum bounding box for the shape, that defines the area available for drawing the contents (such as text).

wxShape::GetBrush

wxBrush* GetBrush() const

Returns the brush used for filling the shape.

wxShape::GetCanvas

wxShapeCanvas* GetCanvas() const

Gets the internal canvas pointer.

wxShape::GetCentreResize

bool GetCentreResize() const

Returns TRUE if the shape is to be resized from the centre (the centre stands still), or FALSE if from the corner or side being dragged (the other corner or side stands still).

wxShape::GetChildren

wxList& GetChildren() const

Returns a reference to the list of children for this shape.

wxShape::GetClientData

wxObject* GetClientData()

Gets the client data associated with the shape (NULL if there is none).

wxShape::GetDisableLabel

bool GetDisableLabel() const

Returns TRUE if the default region will not be shown, FALSE otherwise.

wxShape::GetEventHandler

wxShapeEvtHandler* GetEventHandler() const

Returns the event handler for this shape.

wxShape::GetFixedHeight

bool GetFixedHeight() const

Returns TRUE if the shape cannot be resized in the vertical plane.

wxShape::GetFixedSize

void GetFixedSize(bool * x, bool * y)

Returns flags indicating whether the shape is of fixed size in either direction.

wxShape::GetFixedWidth

bool GetFixedWidth() const

Returns TRUE if the shape cannot be resized in the horizontal plane.

wxShape::GetFont

wxFont* GetFont(int regionId = 0) const

Gets the font for the specified text region.

wxShape::GetFunctor

wxString GetFunctor() const

Gets a string representing the type of the shape, to be used when writing out shape descriptions to a file. This is overridden by each derived shape class to provide an appropriate type string. By default, "node_image" is used for non-line shapes, and "arc_image" for lines.

wxShape::GetId

long GetId() const

Returns the integer identifier for this shape.

wxShape::GetLinePosition

int GetLinePosition(wxLineShape* line)

Gets the zero-based position of *line* in the list of lines for this shape.

wxShape::GetLines

wxList& GetLines() const

Returns a reference to the list of lines connected to this shape.

wxShape::GetMaintainAspectRatio**bool GetMaintainAspectRatio() const**

If returns TRUE, resizing the shape will not change the aspect ratio (width and height will be in the original proportion).

wxShape::GetNumberOfAttachments**int GetNumberOfAttachments() const**

Gets the number of attachment points for this shape.

wxShape::GetNumberOfTextRegions**int GetNumberOfTextRegions() const**

Gets the number of text regions for this shape.

wxShape::GetParent**wxShape * GetParent() const**

Returns the parent of this shape, if it is part of a composite.

wxShape::GetPen**wxPen* GetPen() const**

Returns the pen used for drawing the shape's outline.

wxShape::GetPerimeterPoint**bool GetPerimeterPoint(double x1, double y1, double x2, double y2, double *x3, double *y3)**

Gets the point at which the line from (x1, y1) to (x2, y2) hits the shape. Returns TRUE if the line hits the perimeter.

wxShape::GetRegionId**int GetRegionId(const wxString& name)**

Gets the region's identifier by name. This is *not* unique for within an entire composite, but is unique for the image.

wxShape::GetRegionName**wxString GetRegionName(int regionId = 0)**

Gets the region's name. A region's name can be used to uniquely determine a region within an entire composite image hierarchy. See also *wxShape::SetRegionName* (p. 50).

wxShape::GetRegions**wxList& GetRegions()**

Returns the list of wxShapeRegions.

wxShape::GetRotation**double GetRotation() const**

Returns the angle of rotation in radians.

wxShape::GetSensitivityFilter**void GetSensitivityFilter() const**

Returns the sensitivity filter, a bitlist of values. See *wxShape::SetSensitivityFilter* (p. 50).

wxShape::GetShadowMode**int SetShadowMode() const**

Returns the shadow mode. See *wxShape::SetShadowMode* (p. 50).

wxShape::GetSpaceAttachments**bool GetSpaceAttachments() const**

Indicates whether lines should be spaced out evenly at the point they touch the node (TRUE), or whether they should join at a single point (FALSE).

wxShape::GetTextColour**wxString GetTextColour(int regionId = 0) const**

Gets the colour for the specified text region.

wxShape::GetTopAncestor**wxShape * GetTopAncestor() const**

Returns the top-most ancestor of this shape (the root of the composite).

wxShape::GetX**double GetX() const**

Gets the x position of the centre of the shape.

wxShape::GetY**double GetY() const**

Gets the y position of the centre of the shape.

wxShape::HitTest**bool HitTest(double x, double y, int* attachment, double* distance)**

Given a point on a canvas, returns TRUE if the point was on the shape, and returns the nearest attachment point and distance from the given point and target.

wxShape::Insert**void InsertInCanvas(wxShapeCanvas* canvas)**

Inserts the shape at the front of the shape list of *canvas*.

wxShape::IsHighlighted**bool IsHighlighted() const**

Returns TRUE if the shape is highlighted. Shape highlighting is unimplemented.

wxShape::IsShown**bool IsShown() const**

Returns TRUE if the shape is in a visible state, FALSE otherwise. Note that this has nothing to do with whether the window is hidden or the shape has scrolled off the canvas; it refers to the internal visibility flag.

wxShape::MakeControlPoints**void MakeControlPoints()**

Make a list of control points (draggable handles) appropriate to the shape.

wxShape::MakeMandatoryControlPoints**void MakeMandatoryControlPoints()**

Make the mandatory control points. For example, the control point on a dividing line

should appear even if the divided rectangle shape's handles should not appear (because it is the child of a composite, and children are not resizable).

wxShape::Move

void Move(wxDC& dc, double x1, double y1, bool display = TRUE)

Move the shape to the given position, redrawing if *display* is TRUE.

wxShape::MoveLineToNewAttachment

void MoveLineToNewAttachment(wxDC& dc, wxLineShape* toMove, double x, double y)

Move the given line (which must already be attached to the shape) to a different attachment point on the shape, or a different order on the same attachment.

Calls *wxShape::AttachmentSortTest* (p. 36) and then *wxShapeEvtHandler::OnChangeAttachment* (p. 58).

wxShape::MoveLinks

void MoveLinks(wxDC& dc)

Redraw all the lines attached to the shape.

wxShape::NameRegions

void NameRegions(const wxString& parentName = "")

Make unique names for all the regions in a shape or composite shape.

wxShape::Rotate

void Rotate(double x, double y, double theta)

Rotate about the given axis by the given amount in radians (does nothing for most shapes). But even non-rotating shapes should record their notional rotation in case it's important (e.g. in dog-leg code).

wxShape::ReadConstraints

void ReadConstraints(wxExpr* clause, wxExprDatabase* database)

If the shape is a composite, it may have constraints that need to be read in in a separate pass.

wxShape::ReadAttributes

void ReadAttributes(wxExpr* clause)

Reads the attributes (data member values) from the given expression.

wxShape::ReadRegions**void ReadRegions(wxExpr *clause)**

Reads in the regions.

wxShape::Recentre**void Recentre()**

Does recentring (or other formatting) for all the text regions for this shape.

wxShape::RemoveFromCanvas**void RemoveFromCanvas(wxShapeCanvas* canvas)**

Removes the shape from the canvas.

wxShape::ResetControlPoints**void ResetControlPoints()**

Resets the positions of the control points (for instance when the shape's shape has changed).

wxShape::ResetMandatoryControlPoints**void ResetMandatoryControlPoints()**

Reset the mandatory control points. For example, the control point on a dividing line should appear even if the divided rectangle shape's handles should not appear (because it is the child of a composite, and children are not resizable).

wxShape::Recompute**bool Recompute()**

Recomputes any constraints associated with the shape (normally applicable to wxCompositeShapes only, but harmless for other classes of shape).

wxShape::RemoveLine**void RemoveLine(wxLineShape* line)**

Removes the given line from the shape's list of attached lines.

wxShape::Select

void Select(bool *select* = *TRUE*)

Selects or deselects the given shape, drawing or erasing control points (handles) as necessary.

wxShape::Selected

bool Selected() const

TRUE if the shape is currently selected.

wxShape::SetAttachmentMode

void SetAttachmentMode(bool *flag*)

Sets the attachment mode to TRUE or FALSE. If TRUE, attachment points will be significant when drawing lines to and from this shape. If FALSE, lines will be drawn as if to the centre of the shape.

wxShape::SetBrush

void SetBrush(wxBrush **brush*)

Sets the brush for filling the shape's shape.

wxShape::SetCanvas

void SetCanvas(wxShapeCanvas* *theCanvas*)

Identical to *wxShape::Attach* (p. 48).

wxShape::SetCentreResize

void SetCentreResize(bool *cr*)

Specify whether the shape is to be resized from the centre (the centre stands still) or from the corner or side being dragged (the other corner or side stands still).

wxShape::SetClientData

void SetClientData(wxObject **clientData*)

Sets the client data.

wxShape::SetDefaultRegionSize

void SetDefaultRegionSize()

Set the default region to be consistent with the shape size.

wxShape::SetDisableLabel**void SetDisableLabel(bool flag)**

Set *flag* to TRUE to stop the default region being shown, FALSE otherwise.

wxShape::SetDraggable**void SetDraggable(bool drag, bool recursive = FALSE)**

Sets the shape to be draggable or not draggable.

wxShape::SetDrawHandles**void SetDrawHandles(bool drawH)**

Sets the *drawHandles* flag for this shape and all descendants. If *drawH* is TRUE (the default), any handles (control points) will be drawn. Otherwise, the handles will not be drawn.

wxShape::SetEventHandler**void GetEventHandler(wxShapeEvtHandler *handler)**

Sets the event handler for this shape.

wxShape::SetFixedSize**void SetFixedSize(bool x, bool y)**

Sets the shape to be of the given, fixed size.

wxShape::SetFont**void SetFont(wxFont *font, int regionId = 0)**

Sets the font for the specified text region.

wxShape::SetFormatMode**void SetFormatMode(int mode, int regionId = 0)**

Sets the format mode of the default text region. The argument can be a bit list of the following:

FORMAT_NONE No formatting.

FORMAT_CENTRE_HORIZ Horizontal centring.

FORMAT_CENTRE_VERT Vertical centring.

wxShape::SetHighlight**void SetHighlight**(bool *hi*, bool *recurse* = FALSE)

Sets the highlight for a shape. Shape highlighting is unimplemented.

wxShape::SetId**void SetId**(long *id*)

Set the integer identifier for this shape.

wxShape::SetMaintainAspectRatio**void SetMaintainAspectRatio**(bool *flag*)

If the argument is TRUE, tells the shape that resizes should not change the aspect ratio (width and height should be in the original proportion).

wxShape::SetPen**void SetPen**(wxPen **pen*)

Sets the pen for drawing the shape's outline.

wxShape::SetRegionName**void SetRegionName**(const wxString& *name*, int *regionId* = 0)

Sets the name for this region. The name for a region is unique within the scope of the whole composite, whereas a region id is unique only for a single image.

wxShape::SetSensitivityFilter**void SetSensitivityFilter**(int *sens*=OP_ALL, bool *recursive* = FALSE)

Sets the shape to be sensitive or insensitive to specific mouse operations.

sens is a bitlist of the following:

- OP_CLICK_LEFT
- OP_CLICK_RIGHT
- OP_DRAG_LEFT
- OP_DRAG_RIGHT
- OP_ALL (equivalent to a combination of all the above).

wxShape::SetShadowMode

void SetShadowMode(int mode, bool redraw = FALSE)

Sets the shadow mode (whether a shadow is drawn or not). *mode* can be one of the following:

SHADOW_NONE No shadow (the default).

SHADOW_LEFT Shadow on the left side.

SHADOW_RIGHT Shadow on the right side.

wxShape::SetSize

void SetSize(double x, double y, bool recursive = TRUE)

Sets the shape's size.

wxShape::SetSpaceAttachments

void SetSpaceAttachments(bool sp)

Indicate whether lines should be spaced out evenly at the point they touch the node (TRUE), or whether they should join at a single point (FALSE).

wxShape::SetTextColour

void SetTextColour(const wxString& colour, int regionId = 0)

Sets the colour for the specified text region.

wxShape::SetX

void SetX(double x)

Sets the x position of the shape.

wxShape::SetY

void SetY(double y)

Sets the y position of the shape.

wxShape::SpaceAttachments

void SpaceAttachments(bool sp)

Sets the spacing mode: if TRUE, lines at the same attachment point will be spaced evenly across that side of the shape. If false, all lines at the same attachment point will emanate from the same point.

wxShape::Show

void Show(bool show)

Sets a flag indicating whether the shape should be drawn.

wxShape::Unlink

void Unlink()

If the shape is a line, unlinks the nodes attached to the shape, removing itself from the list of lines for each of the 'to' and 'from' nodes.

wxShape::WriteAttributes

void WriteAttributes(wxExpr *clause)

Writes the shape's attributes (data member values) into the given expression.

wxShape::WriteRegions

void WriteRegions(wxExpr *clause)

Writes the regions.

wxShapeCanvas

A canvas for drawing diagrams on.

Derived from

wxScrolledWindow

See also

wxDiagram (p. 9)

wxShapeCanvas::wxShapeCanvas

wxShapeCanvas(wxWindow* parent = NULL, wxWindowID id = -1, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxBORDER)

Constructor.

wxShapeCanvas::~~wxShapeCanvas

~wxShapeCanvas()

Destructor.

wxShapeCanvas::AddShape**void AddShape(wxShape *shape, wxShape *addAfter = NULL)**

Adds a shape to the diagram. If *addAfter* is non-NULL, the shape will be added after this one.

wxShapeCanvas::FindShape**wxShape * FindShape(double x1, double y, int *attachment, wxClassInfo *info = NULL, wxShape *notImage = NULL)**

Find a shape under this mouse click. Returns the shape (or NULL), and the nearest attachment point.

If *info* is non-NULL, a shape whose class which is a descendant of the desired class is found.

If *notImage* is non-NULL, shapes which are descendants of *notImage* are ignored.

wxShapeCanvas::FindFirstSensitiveShape**wxShape * FindFirstSensitiveShape(double x1, double y, int *attachment, int op)**

Finds the first sensitive shape whose sensitivity filter matches *op*, working up the hierarchy of composites until one (or none) is found.

wxShapeCanvas::GetDiagram**wxDiagram* GetDiagram() const**

Returns the canvas associated with this diagram.

wxShapeCanvas::GetGridSpacing**double GetGridSpacing() const**

Returns the grid spacing.

wxShapeCanvas::GetMouseTolerance**int GetMouseTolerance() const**

Returns the tolerance within which a mouse move is ignored.

wxShapeCanvas::GetShapeList**wxList* GetShapeList() const**

Returns a pointer to the internal shape list.

wxShapeCanvas::GetQuickEditMode**bool GetQuickEditMode() const**

Returns quick edit mode for the associated diagram.

wxShapeCanvas::InsertShape**void InsertShape(wxShape* shape)**

Inserts a shape at the front of the shape list.

wxShapeCanvas::OnBeginDragLeft**void OnBeginDragLeft(double x, double y, int keys = 0)**

Called when the start of a left-button drag event on the canvas background is detected by `OnEvent`. You may override this member; by default it does nothing.

keys is a bit list of the following:

- KEY_SHIFT
- KEY_CTRL

See also `wxShapeCanvas::OnDragLeft` (p. 55), `wxShapeCanvas::OnEndDragLeft` (p. 54).

wxShapeCanvas::OnBeginDragRight**void OnBeginDragRight(double x, double y, int keys = 0)**

Called when the start of a right-button drag event on the canvas background is detected by `OnEvent`. You may override this member; by default it does nothing.

keys is a bit list of the following:

- KEY_SHIFT
- KEY_CTRL

See also `wxShapeCanvas::OnDragRight` (p. 55), `wxShapeCanvas::OnEndDragRight` (p. 55).

wxShapeCanvas::OnEndDragLeft**void OnEndDragLeft(double x, double y, int keys = 0)**

Called when the end of a left-button drag event on the canvas background is detected by `OnEvent`. You may override this member; by default it does nothing.

keys is a bit list of the following:

- KEY_SHIFT
- KEY_CTRL

See also *wxShapeCanvas::OnDragLeft* (p. 55), *wxShapeCanvas::OnBeginDragLeft* (p. 54).

wxShapeCanvas::OnEndDragRight

void OnEndDragRight(double x, double y, int keys = 0)

Called when the end of a right-button drag event on the canvas background is detected by *OnEvent*. You may override this member; by default it does nothing.

keys is a bit list of the following:

- KEY_SHIFT
- KEY_CTRL

See also *wxShapeCanvas::OnDragRight* (p. 55), *wxShapeCanvas::OnBeginDragRight* (p. 54).

wxShapeCanvas::OnDragLeft

void OnDragLeft(bool draw, double x, double y, int keys = 0)

Called when a left-button drag event on the canvas background is detected by *OnEvent*. You may override this member; by default it does nothing.

draw is alternately TRUE and FALSE, to assist drawing and erasing.

keys is a bit list of the following:

- KEY_SHIFT
- KEY_CTRL

See also *wxShapeCanvas::OnBeginDragLeft* (p. 54), *wxShapeCanvas::OnEndDragLeft* (p. 54).

wxShapeCanvas::OnDragRight

void OnDragRight(bool draw, double x, double y, int keys = 0)

Called when a right-button drag event on the canvas background is detected by *OnEvent*. You may override this member; by default it does nothing.

draw is alternately TRUE and FALSE, to assist drawing and erasing.

keys is a bit list of the following:

- KEY_SHIFT

- KEY_CTRL

See also *wxShapeCanvas::OnBeginDragRight* (p. 54), *wxShapeCanvas::OnEndDragRight* (p. 55).

wxShapeCanvas::OnLeftClick

void OnLeftClick(double x, double y, int keys = 0)

Called when a left click event on the canvas background is detected by OnEvent. You may override this member; by default it does nothing.

keys is a bit list of the following:

- KEY_SHIFT
- KEY_CTRL

wxShapeCanvas::OnRightClick

void OnRightClick(double x, double y, int keys = 0)

Called when a right click event on the canvas background is detected by OnEvent. You may override this member; by default it does nothing.

keys is a bit list of the following:

- KEY_SHIFT
- KEY_CTRL

wxShapeCanvas::Redraw

void Redraw()

Calls *wxDiagram::Redraw*.

wxShapeCanvas::RemoveShape

void RemoveShape(wxShape *shape)

Calls *wxDiagram::RemoveShape*.

wxShapeCanvas::SetDiagram

void SetDiagram(wxDiagram *diagram)

Sets the diagram associated with this diagram.

wxShapeCanvas::Snap

void Snap(double *x, double *y)

Calls wxDiagram::Snap.

wxShapeEvtHandler

wxShapeEvtHandler is a class from which wxShape (and therefore all shape classes) are derived. A wxShape also contains a pointer to its current wxShapeEvtHandler. Event handlers can be swapped in and out, altering the behaviour of a shape. This allows, for example, a range of behaviours to be redefined in one class, rather than requiring each shape class to be subclassed.

Derived from

wxObject

wxShapeEvtHandler::m_handlerShape

wxShape* m_handlerShape

Pointer to the shape associated with this handler.

wxShapeEvtHandler::m_previousHandler

wxShapeEvtHandler* m_previousHandler

Pointer to the previous handler.

wxShapeEvtHandler::wxShapeEvtHandler

void wxShapeEvtHandler(wxShapeEvtHandler *previous = NULL, wxShape *shape = NULL)

Constructs a new event handler.

wxShapeEvtHandler::~~wxShapeEvtHandler

void ~wxShapeEvtHandler()

Destructor.

wxShapeEvtHandler::CopyData

void CopyData(wxShapeEvtHandler& handler)

A virtual function to copy the data from this object to *handler*. Override if you derive from wxShapeEvtHandler and have data to copy.

wxShapeEvtHandler::CreateNewCopy**wxShapeEvtHandler* CreateNewCopy()**

Creates a new event handler object of the same class as this object, and then calls *wxShapeEvtHandler::CopyData* (p. 57).

wxShapeEvtHandler::GetPreviousHandler**wxShapeEvtHandler* GetPreviousHandler() const**

Returns the previous handler.

wxShapeEvtHandler::GetShape**wxShape* GetShape() const**

Returns the shape associated with this handler.

wxShapeEvtHandler::OnBeginDragLeft**void OnBeginDragLeft(double x, double y, int keys=0, int attachment = 0)**

Called when the user is beginning to drag using the left mouse button.

wxShapeEvtHandler::OnBeginDragRight**void OnBeginDragRight(double x, double y, int keys=0, int attachment = 0)**

Called when the user is beginning to drag using the right mouse button.

wxShapeEvtHandler::OnBeginSize**void OnBeginSize(double width, double height)**

Called when a shape starts to be resized.

wxShapeEvtHandler::OnChangeAttachment**void OnChangeAttachment(int attachment, wxLineShape* line, wxList& ordering)**

Override this to prevent or intercept line reordering. *wxShape*'s implementation of this function calls *wxShape::ApplyAttachmentOrdering* (p. 36) to apply the new ordering.

wxShapeEvtHandler::OnDragLeft**void OnDragLeft(bool draw, double x, double y, int keys=0, int attachment = 0)**

Called twice when the shape is being dragged, once to allow erasing the old image, and again to allow drawing at the new position.

wxShapeEvtHandler::OnDragRight**void OnDragRight**(bool *draw*, double *x*, double *y*, int *keys*=0, int *attachment* = 0)

Called twice when the shape is being dragged, once to allow erasing the old image, and again to allow drawing at the new position.

wxShapeEvtHandler::OnDraw**void OnDraw**(wxDC& *dc*)

Defined for each class to draw the main graphic, but not the contents.

wxShapeEvtHandler::OnDrawContents**void OnDrawContents**(wxDC& *dc*)

Defined for each class to draw the contents of the shape, such as text.

wxShapeEvtHandler::OnDrawControlPoints**void OnDrawControlPoints**(wxDC& *dc*)

Called when the shape's control points (handles) should be drawn.

wxShapeEvtHandler::OnDrawOutline**void OnDrawOutline**(wxDC& *dc*)

Called when the outline of the shape should be drawn.

wxShapeEvtHandler::OnEndDragLeft**void OnEndDragLeft**(double *x*, double *y*, int *keys*=0, int *attachment* = 0)

Called when the user is stopping dragging using the left mouse button.

wxShapeEvtHandler::OnEndDragRight**void OnEndDragRight**(double *x*, double *y*, int *keys*=0, int *attachment* = 0)

Called when the user is stopping dragging using the right mouse button.

wxShapeEvtHandler::OnEndSize**void OnEndSize**(double *width*, double *height*)

Called after a shape is resized.

wxShapeEvtHandler::OnErase

void OnErase(wxDC& dc)

Called when the whole shape should be erased.

wxShapeEvtHandler::OnEraseContents

void OnEraseContents(wxDC& dc)

Called when the contents should be erased.

wxShapeEvtHandler::OnEraseControlPoints

void OnEraseControlPoints(wxDC& dc)

Called when the shape's control points (handles) should be erased.

wxShapeEvtHandler::OnHighlight

void OnHighlight(wxDC& dc)

Called when the shape should be highlighted.

wxShapeEvtHandler::OnLeftClick

void OnLeftClick(double x, double y, int keys = 0, int attachment = 0)

Called when the shape receives a left mouse click event.

wxShapeEvtHandler::OnMoveLink

void OnMoveLink(wxDC& dc, bool moveControlPoints = TRUE)

Called when the line attached to an shape need to be repositioned, because the shape has moved.

wxShapeEvtHandler::OnMoveLinks

void OnMoveLinks(wxDC& dc)

Called when the lines attached to an shape need to be repositioned, because the shape has moved.

wxShapeEvtHandler::OnMovePost

bool OnMovePost(wxDC& dc, double x, double y, double oldX, double oldY, bool display = TRUE)

Called just after the shape receives a move request.

wxShapeEvtHandler::OnMovePre

bool OnMovePre(wxDC& dc, double x, double y, double oldX, double oldY, bool display = TRUE)

Called just before the shape receives a move request. Returning TRUE allows the move to be processed; returning FALSE vetoes the move.

wxShapeEvtHandler::OnRightClick

void OnRightClick(double x, double y, int keys = 0, int attachment = 0)

Called when the shape receives a mouse click event.

wxShapeEvtHandler::OnSize

void OnSize(double x, double y)

Called when the shape receives a resize request.

wxShapeEvtHandler::OnSizingBeginDragLeft

void OnSizingBeginDragLeft(wxControlPoint* pt, double x, double y, int keys=0, int attachment = 0)

Called when a sizing drag is beginning.

wxShapeEvtHandler::OnSizingDragLeft

void OnSizingDragLeft(wxControlPoint* pt, bool draw, double x, double y, int keys=0, int attachment = 0)

Called when a sizing drag is occurring.

wxShapeEvtHandler::OnSizingEndDragLeft

void OnSizingEndDragLeft(wxControlPoint* pt, double x, double y, int keys=0, int attachment = 0)

Called when a sizing drag is ending.

wxShapeEvtHandler::SetPreviousHandler

void SetPreviousHandler(wxShapeEvtHandler* handler)

Sets the previous handler.

wxShapeEvtHandler::SetShape

void SetShape(wxShape* shape)

Sets the shape for this handler.

wxTextShape

As wxRectangleShape, but only the text is displayed.

Derived from

wxRectangleShape (p. 34)

wxTextShape::wxTextShape

void wxTextShape(double width = 0.0, double height = 0.0)

Constructor.

wxTextShape::~~wxTextShape

void ~wxTextShape()

Destructor.

Functions

These are the OGL functions.

::wxOGLInitialize

void wxOGLInitialize() Initializes OGL.

::wxOGLCleanUp

void wxOGLCleanUp() Cleans up OGL.

Topic overviews

The following sections describe particular topics.

OGL overview

wxShapeCanvas (p. 52), derived from **wxCanvas**, is the drawing area for a number of *wxShape* (p. 35) instances. Everything drawn on a *wxShapeCanvas* is derived from *wxShape*, which provides virtual member functions for redrawing, creating and destroying resize/selection 'handles', movement and erasing behaviour, mouse click behaviour, calculating the bounding box of the shape, linking nodes with arcs, and so on.

The way a client application copes with 'damage' to the canvas is to erase (white out) anything should no longer be displayed, redraw the shape, and then redraw everything on the canvas to repair any damage. If quick edit mode is on for the canvas, the complete should be omitted by OGL and the application.

Selection handles (called control points in the code) are implemented as *wxRectangleShapes*.

Events are passed to shapes by the canvas in a high-level form, for example **OnLeftClick**, **OnBeginDragLeft**, **OnDragLeft**, **OnEndDragLeft**. The canvas decides what is a click and what is a drag, whether it is on a shape or the canvas itself, and (by interrogating the shape) which attachment point the click is associated with.

In order to provide event-handling flexibility, each shapes has an 'event handler' associated with it, which by default is the shape itself (all shapes derive from *wxShapeEvtHandler*). An application can modify the event-handling behaviour simply by plugging a new event handler into the shape. This can avoid the need for multiple inheritance when new properties and behaviour are required for a number of different shape classes: instead of overriding each class, one new event handler class can be defined and used for all existing shape classes.

A range of shapes have been predefined in the library, including rectangles, ellipses, polygons. A client application can derive from these shapes and/or derive entirely new shapes from *wxShape*.

Instances of a class called *wxDiagram* (p. 9) organise collections of shapes, providing default file input and output behaviour.

wxDividedShape overview

Classes: *wxDividedShape* (p. 22)

A *wxDividedShape* is a rectangle with a number of vertical divisions. Each division may have its text formatted with independent characteristics, and the size of each division relative to the whole image may be specified.

Once a `wxDividedShape` has been created, the user may move the divisions with the mouse. By pressing `Ctrl` while right-clicking, the region attributes can be edited.

Here are examples of creating `wxDividedShape` objects:

```
/*
 * Divided rectangle with 3 regions
 *
 */

wxDividedShape *dividedRect = new wxDividedShape(50, 60);

wxShapeRegion *region = new wxShapeRegion;
region->SetProportions(0.0, 0.25);
dividedRect->AddRegion(region);

region = new wxShapeRegion;
region->SetProportions(0.0, 0.5);
dividedRect->AddRegion(region);

region = new wxShapeRegion;
region->SetProportions(0.0, 0.25);
dividedRect->AddRegion(region);

dividedRect->SetSize(50, 60); // Allow it to calculate region
sizes
dividedRect->SetPen(wxBLACK_PEN);
dividedRect->SetBrush(wxWHITE_BRUSH);
dividedRect->Show(TRUE);
dividedRect->NameRegions();

/*
 * Divided rectangle with 3 regions, rounded
 *
 */

wxDividedShape *dividedRect3 = new wxDividedShape(50, 60);
dividedRect3->SetCornerRadius(-0.4);

region = new wxShapeRegion;
region->SetProportions(0.0, 0.25);
dividedRect3->AddRegion(region);

region = new wxShapeRegion;
region->SetProportions(0.0, 0.5);
dividedRect3->AddRegion(region);

region = new wxShapeRegion;
region->SetProportions(0.0, 0.25);
dividedRect3->AddRegion(region);

dividedRect3->SetSize(50, 60); // Allow it to calculate region
sizes
dividedRect3->SetPen(wxBLACK_PEN);
dividedRect3->SetBrush(wxWHITE_BRUSH);
dividedRect3->Show(TRUE);
dividedRect3->NameRegions();
```

wxCompositeShape overview

Classes: *wxCompositeShape* (p. 19), *wxOGLConstraint* (p. 6)

The *wxCompositeShape* allows fairly complex shapes to be created, and maintains a set of constraints which specify the layout and proportions of child shapes.

Add child shapes to a *wxCompositeShape* using *AddChild* (p. 20), and add constraints using *AddConstraint* (p. 20).

After children and shapes have been added, call *Recompute* (p. 21) which will return TRUE if the constraints could be satisfied, FALSE otherwise. If constraints have been correctly and consistently specified, this call will succeed.

If there is more than one child, constraints must be specified: OGL cannot calculate the size and position of children otherwise. Don't assume that children will simply move relative to the parent without the use of constraints.

To specify a constraint, you need three things:

1. a constraint type, such as `gyCONSTRAINT_CENTRED_VERTICALLY`;
2. a reference shape, with respect to which other shapes are going to be positioned - the *constraining* shape;
3. a list of one or more shapes to be constrained: the *constrained* shapes.

The constraining shape can be either the parent of the constrained shapes, or a sibling. The constrained shapes must all be siblings of each other.

For an exhaustive list and description of the available constraint types, see the *wxOGLConstraint constructor* (p. 6). Note that most constraints operate in one dimension only (vertically or horizontally), so you will usually need to specify constraints in pairs.

You can set the spacing between constraining and constrained shapes by calling *wxOGLConstraint::SetSpacing* (p. 8).

Finally, a *wxCompositeShape* can have *divisions*, which are special child shapes of class *wxDivisionShape* (not to be confused with *wxDividedShape*). The purpose of this is to allow the composite to be divided into user-adjustable regions (divisions) into which other shapes can be dropped dynamically, given suitable application code. Divisions allow the child shapes to have an identity of their own - they can be manipulated independently of their container - but to behave as if they are contained with the division, moving with the parent shape. Divisions boundaries can themselves be moved using the mouse.

To create an initial division, call *wxCompositeShape::MakeContainer* (p. 21). Make further divisions by calling *wxDivisionShape::Divide* (p. 23).

Bugs

These are the known bugs.

- In the OGLEdit sample, .dia files are output double-spaced due to an unidentified bug in the way a stream is converted to a file.

Change log

Version 3.0, September 8th 1998

- Version for wxWindows 2.0.
- Various enhancements especially to wxDrawnShape (multiple metafiles, for different orientations).
- More ability to override functions e.g. OnSizeDragLeft, so events can be intercepted for Do/Undo.

Version 2.0, June 1st 1996

- First publicly released version.

Index

—:—

::wxOGLCleanUp, 62
::wxOGLInitialize, 62

—~—

~wxBitmapShape, 9
~wxCircleShape, 19
~wxCompositeShape, 20
~wxDiagram, 10
~wxDividedShape, 22
~wxDivisionShape, 23
~wxDrawnShape, 15
~wxEllipseShape, 27
~wxLineShape, 27
~wxOGLConstraint, 8
~wxPolygonShape, 33
~wxRectangleShape, 34
~wxShape, 35
~wxShapeCanvas, 52
~wxShapeEvtHandler, 57
~wxTextShape, 62

—A—

AddArrow, 27
AddArrowOrdered, 28
AddChild, 20
AddConstraint, 20
AddLine, 35
AddPolygonPoint, 33
AddRegion, 35
AddShape, 10, 53
AddText, 36
AddToCanvas, 36
AdjustBottom, 23
AdjustLeft, 23
AdjustRight, 23
AdjustTop, 23
AncestorSelected, 36
ApplyAttachmentOrdering, 36
AssignNewIds, 36
Attach, 36
AttachmentIsValid, 36
AttachmentSortTest, 36

—C—

CalcSimpleAttachment, 37
CalculatePolygonCentre, 34
CalculateSize, 15, 20, 37
Clear, 10
ClearArrow, 28

ClearArrowsAtPosition, 28
ClearAttachments, 37
ClearRegions, 38
ClearText, 38
Constrain, 38
Copy, 38
CopyData, 57
Create, 33
CreateNewCopy, 38, 58

—D—

DeleteArrowHead, 29
DeleteConstraint, 20
DeleteConstraintsInvolvingChild, 20
DeleteControlPoints, 39
DeleteLineControlPoint, 29
DeletePolygonPoint, 34
DeletesAllShapes, 10
DestroyClippingRect, 15
Detach, 39
Divide, 23
Draggable, 39
Draw, 39
DrawArc, 15
DrawArrow, 29
DrawArrows, 29
DrawAtAngle, 15
DrawContents, 39
DrawEllipticArc, 15
DrawLine, 16
DrawLines, 16
DrawLinks, 39
DrawOutline, 10
DrawPoint, 16
DrawPolygon, 16
DrawRectangle, 16
DrawRegion, 29
DrawRoundedRectangle, 16
DrawSpline, 16
DrawText, 16

—E—

EditEdge, 24
EditRegions, 22
Equals, 8
Erase, 39
EraseContents, 39
EraseLinks, 40
EraseRegion, 29
Evaluate, 8

—F—

FindArrowHead, 29
FindConstraint, 21
FindContainerImage, 20, 21
FindFirstSensitiveShape, 53
FindLineEndPoints, 29
FindLinePosition, 30
FindMinimumWidth, 30
FindNth, 30
FindRegion, 40
FindRegionNames, 40
FindShape, 10, 53
Flash, 40
FormatText, 40

—G—

GetAngle, 17
GetAttachmentFrom, 30
GetAttachmentMode, 40
GetAttachmentPosition, 40
GetAttachmentTo, 30
GetBitmap, 9
GetBottomSide, 24
GetBoundingBoxMax, 41
GetBoundingBoxMin, 41
GetBrush, 41
GetCanvas, 10, 41
GetCentreResize, 41
GetChildren, 41
GetClientData, 41
GetConstraints, 21
GetCount, 11
GetDiagram, 53
GetDisableLabel, 41
GetDivisions, 21
GetEnds, 30
GetEventHandler, 42, 49
GetFilename, 9
GetFixedHeight, 42
GetFixedSize, 42
GetFixedWidth, 42
GetFont, 42
GetFrom, 30
GetFunctor, 42
GetGridSpacing, 11, 53
GetHandleSide, 24
GetId, 42
GetLabelPosition, 30
GetLeftSide, 24
GetLeftSideColour, 24
GetLeftSidePen, 24
GetLinePosition, 42
GetLines, 43
GetMaintainAspectRatio, 43
GetMetaFile, 17
GetMouseTolerance, 11, 53
GetNextControlPoint, 31
GetNumberOfAttachments, 43
GetNumberOfTextRegions, 43
GetParent, 43

GetPen, 43
GetPerimeterPoint, 43
GetPoints, 34
GetPreviousHandler, 58
GetQuickEditMode, 11, 54
GetRegionId, 43
GetRegionName, 44
GetRegions, 44
GetRightSide, 24
GetRotatation, 44
GetRotation, 17
GetSensitivityFilter, 44
GetShape, 58
GetShapeList, 11, 53
GetSnapToGrid, 11
GetSpaceAttachments, 44
GetTextColour, 44
GetTo, 31
GetTopAncestor, 44
GetTopSide, 24
GetTopSideColour, 25
GetTopSidePen, 25
GetX, 45
GetY, 45

—H—

HitTest, 45

—I—

Initialise, 31
InsertInCanvas, 45
InsertLineControlPoint, 31
InsertShape, 11, 54
IsEnd, 31
IsHighlighted, 45
IsShown, 45
IsSpline, 31

—L—

LoadFile, 11
LoadFromMetaFile, 17

—M—

m_handlerShape, 57
m_previousHandler, 57
MakeContainer, 21
MakeControlPoints, 45
MakeLineControlPoints, 31
MakeMandatoryControlPoints, 45
Move, 46
MoveLineToNewAttachment, 46
MoveLinks, 46

—N—

NameRegions, 46

—O—

OnBeginDragLeft, 54, 58
 OnBeginDragRight, 54, 58
 OnBeginSize, 58
 OnChangeAttachment, 58
 OnCreateDivision, 21
 OnDatabaseLoad, 12
 OnDatabaseSave, 12
 OnDragLeft, 55, 58
 OnDragRight, 55, 59
 OnDraw, 59
 OnDrawContents, 59
 OnDrawControlPoints, 59
 OnDrawOutline, 59
 OnEndDragLeft, 54, 59
 OnEndDragRight, 55, 59
 OnEndSize, 59
 OnErase, 60
 OnEraseContents, 60
 OnEraseControlPoints, 60
 OnHeaderLoad, 12
 OnHeaderSave, 12
 OnHighlight, 60
 OnLeftClick, 56, 60
 OnMoveLink, 31, 60
 OnMoveLinks, 60
 OnMovePost, 60
 OnMovePre, 61
 OnRightClick, 56, 61
 OnShapeLoad, 12
 OnShapeSave, 12
 OnSize, 61
 OnSizingBeginDragLeft, 61
 OnSizingDragLeft, 61
 OnSizingEndDragLeft, 61

—P—

PopupMenu, 25

—R—

ReadAttributes, 46
 ReadConstraints, 46
 ReadContainerGeometry, 12
 ReadLines, 13
 ReadNodes, 13
 ReadRegions, 47
 Recentre, 47
 RecentreAll, 13
 Recompute, 21, 47
 Redraw, 13, 56
 RemoveAllShapes, 13
 RemoveChild, 21
 RemoveFromCanvas, 47
 RemoveLine, 47
 RemoveShape, 13, 56
 ResetControlPoints, 47
 ResetMandatoryControlPoints, 47
 ResizeAdjoining, 25
 Rotate, 17, 46

—S—

SaveFile, 13
 Scale, 18
 Select, 48
 Selected, 48
 SetAttachmentMode, 48
 SetAttachments, 32
 SetAttachmentTo, 32
 SetBitmap, 9
 SetBottomSide, 25
 SetBrush, 48
 SetCanvas, 13, 48
 SetCentreResize, 48
 SetClientData, 48
 SetClippingRect, 17
 SetCornerRadius, 34
 SetDefaultRegionSize, 48
 SetDiagram, 56
 SetDisableLabel, 49
 SetDraggable, 49
 SetDrawHandles, 49
 SetDrawnBackgroundColour, 18
 SetDrawnBackgroundMode, 18
 SetDrawnBrush, 18
 SetDrawnFont, 18
 SetDrawnPen, 18
 SetDrawnTextColour, 18
 SetEnds, 32
 SetFilename, 9
 SetFixedSize, 49
 SetFont, 49
 SetFormatMode, 49
 SetFrom, 32
 SetGridSpacing, 14
 SetHandleSide, 25
 SetHighlight, 50
 SetId, 50
 SetIgnoreOffsets, 32
 SetLeftSide, 25
 SetLeftSideColour, 26
 SetLeftSidePen, 26
 SetMaintainAspectRatio, 50
 SetMouseTolerance, 14
 SetPen, 50
 SetPreviousHandler, 61
 SetQuickEditMode, 14
 SetRegionName, 50
 SetRegionSizes, 22
 SetRightSide, 26
 SetSaveToFile, 18
 SetSensitivityFilter, 50
 SetShadowMode, 44, 51
 SetShape, 61
 SetSize, 51
 SetSnapToGrid, 14
 SetSpaceAttachments, 51
 SetSpacing, 8
 SetSpline, 32
 SetTextColour, 51
 SetTo, 32
 SetTopSide, 26

SetTopSideColour, 26
 SetTopSidePen, 26
 SetX, 51
 SetY, 51
 Show, 52
 ShowAll, 14
 Snap, 14, 57
 SpaceAttachments, 51
 Straighten, 33

—T—

Translate, 19

—U—

Unlink, 33, 52
 UpdateOriginalPoints, 34

—W—

WriteAttributes, 52
 WriteRegions, 52
 wxBitmapShape, 9
 wxBitmapShape::~~wxBitmapShape, 9
 wxBitmapShape::GetBitmap, 9
 wxBitmapShape::GetFilename, 9
 wxBitmapShape::SetBitmap, 9
 wxBitmapShape::SetFilename, 9
 wxBitmapShape::wxBitmapShape, 9
 wxCircleShape, 19
 wxCircleShape::~~wxCircleShape, 19
 wxCircleShape::wxCircleShape, 19
 wxCompositeShape, 19
 wxCompositeShape::~~wxCompositeShape, 20
 wxCompositeShape::AddChild, 20
 wxCompositeShape::AddConstraint, 20
 wxCompositeShape::CalculateSize, 20
 wxCompositeShape::ContainsDivision, 20
 wxCompositeShape::DeleteConstraint, 20
 wxCompositeShape::DeleteConstraintsInvolving
 Child, 20
 wxCompositeShape::FindConstraint, 21
 wxCompositeShape::FindContainerImage, 21
 wxCompositeShape::GetConstraints, 21
 wxCompositeShape::GetDivisions, 21
 wxCompositeShape::MakeContainer, 21
 wxCompositeShape::OnCreateDivision, 21
 wxCompositeShape::Recompute, 21
 wxCompositeShape::RemoveChild, 21
 wxCompositeShape::wxCompositeShape, 19
 wxDiagram, 10
 wxDiagram::~~wxDiagram, 10
 wxDiagram::AddShape, 10
 wxDiagram::Clear, 10
 wxDiagram::DeleteAllShapes, 10
 wxDiagram::DrawOutline, 10
 wxDiagram::FindShape, 10
 wxDiagram::GetCanvas, 10
 wxDiagram::GetCount, 11
 wxDiagram::GetGridSpacing, 11
 wxDiagram::GetMouseTolerance, 11

wxDiagram::GetQuickEditMode, 11
 wxDiagram::GetShapeList, 11
 wxDiagram::GetSnapToGrid, 11
 wxDiagram::InsertShape, 11
 wxDiagram::LoadFile, 11
 wxDiagram::OnDatabaseLoad, 11
 wxDiagram::OnDatabaseSave, 12
 wxDiagram::OnHeaderLoad, 12
 wxDiagram::OnHeaderSave, 12
 wxDiagram::OnShapeLoad, 12
 wxDiagram::OnShapeSave, 12
 wxDiagram::ReadContainerGeometry, 12
 wxDiagram::ReadLines, 13
 wxDiagram::ReadNodes, 13
 wxDiagram::RecentreAll, 13
 wxDiagram::Redraw, 13
 wxDiagram::RemoveAllShapes, 13
 wxDiagram::RemoveShape, 13
 wxDiagram::SaveFile, 13
 wxDiagram::SetCanvas, 13
 wxDiagram::SetGridSpacing, 13
 wxDiagram::SetMouseTolerance, 14
 wxDiagram::SetQuickEditMode, 14
 wxDiagram::SetSnapToGrid, 14
 wxDiagram::ShowAll, 14
 wxDiagram::Snap, 14
 wxDiagram::wxDiagram, 10
 wxDividedShape, 22
 wxDividedShape::~~wxDividedShape, 22
 wxDividedShape::EditRegions, 22
 wxDividedShape::SetRegionSizes, 22
 wxDividedShape::wxDividedShape, 22
 wxDivisionShape, 23
 wxDivisionShape::~~wxDivisionShape, 23
 wxDivisionShape::AdjustBottom, 23
 wxDivisionShape::AdjustLeft, 23
 wxDivisionShape::AdjustRight, 23
 wxDivisionShape::AdjustTop, 23
 wxDivisionShape::Divide, 23
 wxDivisionShape::EditEdge, 24
 wxDivisionShape::GetBottomSide, 24
 wxDivisionShape::GetHandleSide, 24
 wxDivisionShape::GetLeftSide, 24
 wxDivisionShape::GetLeftSideColour, 24
 wxDivisionShape::GetLeftSidePen, 24
 wxDivisionShape::GetRightSide, 24
 wxDivisionShape::GetTopSide, 24
 wxDivisionShape::GetTopSideColour, 24
 wxDivisionShape::GetTopSidePen, 25
 wxDivisionShape::PopupMenu, 25
 wxDivisionShape::ResizeAdjoining, 25
 wxDivisionShape::SetBottomSide, 25
 wxDivisionShape::SetHandleSide, 25
 wxDivisionShape::SetLeftSide, 25
 wxDivisionShape::SetLeftSideColour, 26
 wxDivisionShape::SetLeftSidePen, 26
 wxDivisionShape::SetRightSide, 26
 wxDivisionShape::SetTopSide, 26
 wxDivisionShape::SetTopSideColour, 26
 wxDivisionShape::SetTopSidePen, 26
 wxDivisionShape::wxDivisionShape, 23
 wxDrawnShape, 15

wxDrawnShape::~~wxDrawnShape, 15
wxDrawnShape::CalculateSize, 15
wxDrawnShape::DestroyClippingRect, 15
wxDrawnShape::DrawArc, 15
wxDrawnShape::DrawAtAngle, 15
wxDrawnShape::DrawEllipticArc, 15
wxDrawnShape::DrawLine, 16
wxDrawnShape::DrawLines, 16
wxDrawnShape::DrawPoint, 16
wxDrawnShape::DrawPolygon, 16
wxDrawnShape::DrawRectangle, 16
wxDrawnShape::DrawRoundedRectangle, 16
wxDrawnShape::DrawSpline, 16
wxDrawnShape::DrawText, 16
wxDrawnShape::GetAngle, 17
wxDrawnShape::GetMetaFile, 17
wxDrawnShape::GetRotation, 17
wxDrawnShape::LoadFromMetaFile, 17
wxDrawnShape::Rotate, 17
wxDrawnShape::Scale, 18
wxDrawnShape::SetClippingRect, 17
wxDrawnShape::SetDrawnBackgroundColour, 17
wxDrawnShape::SetDrawnBackgroundMode, 18
wxDrawnShape::SetDrawnBrush, 18
wxDrawnShape::SetDrawnFont, 18
wxDrawnShape::SetDrawnPen, 18
wxDrawnShape::SetDrawnTextColour, 18
wxDrawnShape::SetSaveToFile, 18
wxDrawnShape::Translate, 19
wxDrawnShape::wxDrawnShape, 15
wxEllipseShape, 27
wxEllipseShape::~~wxEllipseShape, 27
wxEllipseShape::wxEllipseShape, 26
wxLineShape, 27
wxLineShape::~~wxLineShape, 27
wxLineShape::AddArrow, 27
wxLineShape::AddArrowOrdered, 28
wxLineShape::ClearArrow, 28
wxLineShape::ClearArrowsAtPosition, 28
wxLineShape::DeleteArrowHead, 29
wxLineShape::DeleteLineControlPoint, 29
wxLineShape::DrawArrow, 28
wxLineShape::DrawArrows, 29
wxLineShape::DrawRegion, 29
wxLineShape::EraseRegion, 29
wxLineShape::FindArrowHead, 29
wxLineShape::FindLineEndPoints, 29
wxLineShape::FindLinePosition, 30
wxLineShape::FindMinimumWidth, 30
wxLineShape::FindNth, 30
wxLineShape::GetAttachmentFrom, 30
wxLineShape::GetAttachmentTo, 30
wxLineShape::GetEnds, 30
wxLineShape::GetFrom, 30
wxLineShape::GetLabelPosition, 30
wxLineShape::GetNextControlPoint, 31
wxLineShape::GetTo, 31
wxLineShape::Initialise, 31
wxLineShape::InsertLineControlPoint, 31
wxLineShape::IsEnd, 31
wxLineShape::IsSpline, 31
wxLineShape::MakeLineControlPoints, 31
wxLineShape::OnMoveLink, 31
wxLineShape::SetAttachmentFrom, 32
wxLineShape::SetAttachments, 32
wxLineShape::SetAttachmentTo, 32
wxLineShape::SetEnds, 32
wxLineShape::SetFrom, 32
wxLineShape::SetIgnoreOffsets, 32
wxLineShape::SetSpline, 32
wxLineShape::SetTo, 32
wxLineShape::Straighten, 33
wxLineShape::Unlink, 33
wxLineShape::wxLineShape, 27
wxOGLCleanup, 62
wxOGLConstraint, 6
wxOGLConstraint::~~wxOGLConstraint, 8
wxOGLConstraint::Equals, 8
wxOGLConstraint::Evaluate, 8
wxOGLConstraint::SetSpacing, 8
wxOGLConstraint::wxOGLConstraint, 6
wxOGLInitialize, 62
wxPolygonShape, 33
wxPolygonShape::~~wxPolygonShape, 33
wxPolygonShape::AddPolygonPoint, 33
wxPolygonShape::CalculatePolygonCentre, 33
wxPolygonShape::Create, 33
wxPolygonShape::DeletePolygonPoint, 34
wxPolygonShape::GetPoints, 34
wxPolygonShape::UpdateOriginalPoints, 34
wxPolygonShape::wxPolygonShape, 33
wxRectangleShape, 34
wxRectangleShape::~~wxRectangleShape, 34
wxRectangleShape::SetCornerRadius, 34
wxRectangleShape::wxRectangleShape, 34
wxShape, 35
wxShape::~~wxShape, 35
wxShape::AddLine, 35
wxShape::AddRegion, 35
wxShape::AddText, 36
wxShape::AddToCanvas, 36
wxShape::AncestorSelected, 36
wxShape::ApplyAttachmentOrdering, 36
wxShape::AssignNewIds, 36
wxShape::Attach, 36
wxShape::AttachmentIsValid, 36
wxShape::AttachmentSortTest, 36
wxShape::CalcSimpleAttachment, 37
wxShape::CalculateSize, 37
wxShape::ClearAttachments, 37
wxShape::ClearRegions, 38
wxShape::ClearText, 38
wxShape::Constrain, 38
wxShape::Copy, 38
wxShape::CreateNewCopy, 38
wxShape::DeleteControlPoints, 39
wxShape::Detach, 39
wxShape::Draggable, 39
wxShape::Draw, 39
wxShape::DrawContents, 39
wxShape::DrawLinks, 39
wxShape::Erase, 39
wxShape::EraseContents, 39
wxShape::EraseLinks, 40

wxShape::FindRegion, 40
wxShape::FindRegionNames, 40
wxShape::Flash, 40
wxShape::FormatText, 40
wxShape::GetAttachmentMode, 40
wxShape::GetAttachmentPosition, 40
wxShape::GetBoundingBoxMax, 40
wxShape::GetBoundingBoxMin, 41
wxShape::GetBrush, 41
wxShape::GetCanvas, 41
wxShape::GetCentreResize, 41
wxShape::GetChildren, 41
wxShape::GetClientData, 41
wxShape::GetDisableLabel, 41
wxShape::GetEventHandler, 41
wxShape::GetFixedHeight, 42
wxShape::GetFixedSize, 42
wxShape::GetFixedWidth, 42
wxShape::GetFont, 42
wxShape::GetFunctor, 42
wxShape::GetId, 42
wxShape::GetLinePosition, 42
wxShape::GetLines, 42
wxShape::GetMaintainAspectRatio, 43
wxShape::GetNumberOfAttachments, 43
wxShape::GetNumberOfTextRegions, 43
wxShape::GetParent, 43
wxShape::GetPen, 43
wxShape::GetPerimeterPoint, 43
wxShape::GetRegionId, 43
wxShape::GetRegionName, 44
wxShape::GetRegions, 44
wxShape::GetRotation, 44
wxShape::GetSensitivityFilter, 44
wxShape::GetShadowMode, 44
wxShape::GetSpaceAttachments, 44
wxShape::GetTextColour, 44
wxShape::GetTopAncestor, 44
wxShape::GetX, 45
wxShape::GetY, 45
wxShape::HitTest, 45
wxShape::Insert, 45
wxShape::IsHighlighted, 45
wxShape::IsShown, 45
wxShape::MakeControlPoints, 45
wxShape::MakeMandatoryControlPoints, 45
wxShape::Move, 46
wxShape::MoveLineToNewAttachment, 46
wxShape::MoveLinks, 46
wxShape::NameRegions, 46
wxShape::ReadAttributes, 46
wxShape::ReadConstraints, 46
wxShape::ReadRegions, 47
wxShape::Recentre, 47
wxShape::Recompute, 47
wxShape::RemoveFromCanvas, 47
wxShape::RemoveLine, 47
wxShape::ResetControlPoints, 47
wxShape::ResetMandatoryControlPoints, 47
wxShape::Rotate, 46
wxShape::Select, 47
wxShape::Selected, 48
wxShape::SetAttachmentMode, 48
wxShape::SetBrush, 48
wxShape::SetCanvas, 48
wxShape::SetCentreResize, 48
wxShape::SetClientData, 48
wxShape::SetDefaultRegionSize, 48
wxShape::SetDisableLabel, 49
wxShape::SetDraggable, 49
wxShape::SetDrawHandles, 49
wxShape::SetEventHandler, 49
wxShape::SetFixedSize, 49
wxShape::SetFont, 49
wxShape::SetFormatMode, 49
wxShape::SetHighlight, 50
wxShape::SetId, 50
wxShape::SetMaintainAspectRatio, 50
wxShape::SetPen, 50
wxShape::SetRegionName, 50
wxShape::SetSensitivityFilter, 50
wxShape::SetShadowMode, 50
wxShape::SetSize, 51
wxShape::SetSpaceAttachments, 51
wxShape::SetTextColour, 51
wxShape::SetX, 51
wxShape::Show, 51
wxShape::SpaceAttachments, 51
wxShape::Unlink, 52
wxShape::WriteAttributes, 52
wxShape::WriteRegions, 52
wxShape::wxShape, 35
wxShapeCanvas, 52
wxShapeCanvas::~wxShapeCanvas, 52
wxShapeCanvas::AddShape, 53
wxShapeCanvas::FindFirstSensitiveShape, 53
wxShapeCanvas::FindShape, 53
wxShapeCanvas::GetDiagram, 53
wxShapeCanvas::GetGridSpacing, 53
wxShapeCanvas::GetMouseTolerance, 53
wxShapeCanvas::GetQuickEditMode, 54
wxShapeCanvas::GetShapeList, 53
wxShapeCanvas::InsertShape, 54
wxShapeCanvas::OnBeginDragLeft, 54
wxShapeCanvas::OnBeginDragRight, 54
wxShapeCanvas::OnDragLeft, 55
wxShapeCanvas::OnDragRight, 55
wxShapeCanvas::OnEndDragLeft, 54
wxShapeCanvas::OnEndDragRight, 55
wxShapeCanvas::OnLeftClick, 56
wxShapeCanvas::OnRightClick, 56
wxShapeCanvas::Redraw, 56
wxShapeCanvas::RemoveShape, 56
wxShapeCanvas::SetDiagram, 56
wxShapeCanvas::Snap, 56
wxShapeCanvas::wxShapeCanvas, 52
wxShapeEvtHandler, 57
wxShapeEvtHandler::~wxShapeEvtHandler, 57
wxShapeEvtHandler::CopyData, 57
wxShapeEvtHandler::CreateNewCopy, 58
wxShapeEvtHandler::GetPreviousHandler, 58
wxShapeEvtHandler::GetShape, 58
wxShapeEvtHandler::m_handlerShape, 57
wxShapeEvtHandler::m_previousHandler, 57

wxShapeEvtHandler::OnBeginDragLeft, 58
wxShapeEvtHandler::OnBeginDragRight, 58
wxShapeEvtHandler::OnBeginSize, 58
wxShapeEvtHandler::OnChangeAttachment, 58
wxShapeEvtHandler::OnDragLeft, 58
wxShapeEvtHandler::OnDragRight, 59
wxShapeEvtHandler::OnDraw, 59
wxShapeEvtHandler::OnDrawContents, 59
wxShapeEvtHandler::OnDrawControlPoints, 59
wxShapeEvtHandler::OnDrawOutline, 59
wxShapeEvtHandler::OnEndDragLeft, 59
wxShapeEvtHandler::OnEndDragRight, 59
wxShapeEvtHandler::OnEndSize, 59
wxShapeEvtHandler::OnErase, 59
wxShapeEvtHandler::OnEraseContents, 60
wxShapeEvtHandler::OnEraseControlPoints, 60
wxShapeEvtHandler::OnHighlight, 60
wxShapeEvtHandler::OnLeftClick, 60
wxShapeEvtHandler::OnMoveLink, 60
wxShapeEvtHandler::OnMoveLinks, 60
wxShapeEvtHandler::OnMovePost, 60
wxShapeEvtHandler::OnMovePre, 60
wxShapeEvtHandler::OnRightClick, 61
wxShapeEvtHandler::OnSize, 61
wxShapeEvtHandler::OnSizingBeginDragLeft, 61
wxShapeEvtHandler::OnSizingDragLeft, 61
wxShapeEvtHandler::OnSizingEndDragLeft, 61
wxShapeEvtHandler::SetPreviousHandler, 61
wxShapeEvtHandler::SetShape, 61
wxShapeEvtHandler::wxShapeEvtHandler, 57
wxTextShape, 62
wxTextShape::~wxTextShape, 62
wxTextShape::wxTextShape, 62