

#\$+K!

Object Graphics Library 3.0

Julian Smart

September 1998

Contents

[Introduction](#)

[OGLEdit: a sample OGL application](#)

[Class reference](#)

[Topic overviews](#)

[Bugs](#)

[Change log](#)

^Contents

^Contents

^browse00001

^K Contents

^DisableButton("Up")

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](#) for a general description how the object library works. For details, please see the [class reference](#).

File structure

Introduction

topic0

browse00002

^K Introduction

^DisableButton("Up")

^{\$\$+K!}**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.

{bmc ogledit.bmp}

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

How OGLEdit works

Possible enhancements

^oGLEdit: a sample OGL application

^ogledit

^browse00004

^K OGLEdit a sample OGL application

^DisableButton("Up")

Class reference

These are the main OGL classes.

[wxOGLConstraint](#)
[wxBitmapShape](#)
[wxDiagram](#)
[wxDrawnShape](#)
[wxCircleShape](#)
[wxCompositeShape](#)
[wxDividedShape](#)
[wxDivisionShape](#)
[wxEllipseShape](#)
[wxLineShape](#)
[wxPolygonShape](#)
[wxRectangleShape](#)
[wxPseudoMetaFile](#)
[wxShape](#)
[wxShapeCanvas](#)
[wxShapeEvtHandler](#)
[wxTextShape](#)
[Functions](#)

^Class reference

^Classref

^browse00008

^K Class reference

^DisableButton("Up")

^{\$#+K!}Topic overviews

The following sections describe particular topics.

[OGL overview](#)

[wxDividedShape overview](#)

[wxCompositeShape overview](#)

^Topic overviews

^topic295

^browse00393

^K Topic overviews

^DisableButton("Up")

Bugs

These are the known bugs.

{bmc bullet.bmp} 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.

Bugs

bugs

rowse00397

Bugs

isableButton("Up")

Change log

Version 3.0, September 8th 1998

{bmc bullet.bmp} Version for wxWindows 2.0.

{bmc bullet.bmp} Various enhancements especially to wxDrawnShape (multiple metafiles, for different orientations).

{bmc bullet.bmp} More ability to override functions e.g. OnSizeDragLeft, so events can be intercepted for Do/Undo.

Version 2.0, June 1st 1996

{bmc bullet.bmp} First publicly released version.

Change log

topic296

browse00398

Change log

DisableButton("Up")

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.

File structure

topic1

browse00003

K File structure

enableButton("Up");ChangeButtonBinding("Up", "JumpId('ogl.hlp', `topic0')")

mfutils.h Metafile utilities header.

mfutils.cpp Metafile utilities implementation.

OGLEdit files

OGLEdit comprises the following source files.

{bmc bullet.bmp} doc.h, doc.cpp: MyDiagram, DiagramDocument, DiagramCommand, MyEvtHandler classes related to diagram functionality and documents.

{bmc bullet.bmp} view.h, view.cpp: MyCanvas, DiagramView classes related to visualisation of the diagram.

{bmc bullet.bmp} ogledit.h, ogledit.cpp: MyFrame, MyApp classes related to the overall application.

{bmc bullet.bmp} palette.h, palette.cpp: EditorToolPalette implementing the shape palette.

^oGLEdit files

^topic2

^browse00005

^K OGLEdit files

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `ogledit')")

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.

How OGLEdit works

topic3

browse00006

^K How OGLEdit works

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('ogl.hlp', 'ogledit')")

^{\$\$\$K!}**Possible enhancements**

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

{bmc bullet.bmp} attachment points (arcs are drawn to particular points on a shape)

{bmc bullet.bmp} metafile and bitmaps shapes

{bmc bullet.bmp} divided rectangles

{bmc bullet.bmp} composite shapes, and constraints

{bmc bullet.bmp} creating labels in shape regions

{bmc bullet.bmp} arc labels (OGL has support for three movable labels per arc)

{bmc bullet.bmp} spline and multiple-segment line arcs

{bmc bullet.bmp} adding annotations to node and arc shapes

{bmc bullet.bmp} 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.

^Possible enhancements

^topic4

^browse00007

^K Possible enhancements

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `ogledit')")

`wxOGLConstraint`

[wxCompositeShape overview](#)

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

Derived from

`wxObject`

See also

[wxCompositeShape](#)

Members

[wxOGLConstraint::wxOGLConstraint](#)
[wxOGLConstraint::~~wxOGLConstraint](#)
[wxOGLConstraint::Equals](#)
[wxOGLConstraint::Evaluate](#)
[wxOGLConstraint::SetSpacing](#)

`wxOGLConstraint`

`wxoglconstraint`

`rowse00009`

`wxOGLConstraint`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp', `classref)")`

`$#+K!` **wxBitmapShape**

Draws a bitmap (non-resizable).

Derived from

wxRectangleShape

Members

wxBitmapShape::wxBitmapShape

wxBitmapShape::~~wxBitmapShape

wxBitmapShape::GetBitmap

wxBitmapShape::GetFilename

wxBitmapShape::SetBitmap

wxBitmapShape::SetFilename

`wxBitmapShape`

`wxbitmapshape`

`browse00015`

`K wxBitmapShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `classref')")`

\$#+K! **wxDiagram**

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

Derived from

wxObject

See also

[wxShapeCanvas](#)

Members

[wxDiagram::wxDiagram](#)
[wxDiagram::~~wxDiagram](#)
[wxDiagram::AddShape](#)
[wxDiagram::Clear](#)
[wxDiagram::DeleteAllShapes](#)
[wxDiagram::DrawOutline](#)
[wxDiagram::FindShape](#)
[wxDiagram::GetCanvas](#)
[wxDiagram::GetCount](#)
[wxDiagram::GetGridSpacing](#)
[wxDiagram::GetMouseTolerance](#)
[wxDiagram::GetShapeList](#)
[wxDiagram::GetQuickEditMode](#)
[wxDiagram::GetSnapToGrid](#)
[wxDiagram::InsertShape](#)
[wxDiagram::LoadFile](#)
[wxDiagram::OnDatabaseLoad](#)
[wxDiagram::OnDatabaseSave](#)
[wxDiagram::OnHeaderLoad](#)
[wxDiagram::OnHeaderSave](#)
[wxDiagram::OnShapeLoad](#)
[wxDiagram::OnShapeSave](#)
[wxDiagram::ReadContainerGeometry](#)
[wxDiagram::ReadLines](#)
[wxDiagram::ReadNodes](#)
[wxDiagram::RecentreAll](#)
[wxDiagram::Redraw](#)
[wxDiagram::RemoveAllShapes](#)
[wxDiagram::RemoveShape](#)

^wxDiagram

^wxdiagram

^browse00022

^K wxDiagram

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('ogl.hlp', `classref')")

wxDiagram::SaveFile
wxDiagram::SetCanvas
wxDiagram::SetGridSpacing
wxDiagram::SetMouseTolerance
wxDiagram::SetQuickEditMode
wxDiagram::SetSnapToGrid
wxDiagram::ShowAll
wxDiagram::Snap

`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`](#)

See also [`wxRectangleShape`](#).

Members

[`wxDrawnShape::wxDrawnShape`](#)
[`wxDrawnShape::~~wxDrawnShape`](#)
[`wxDrawnShape::CalculateSize`](#)
[`wxDrawnShape::DestroyClippingRect`](#)
[`wxDrawnShape::DrawArc`](#)
[`wxDrawnShape::DrawAtAngle`](#)
[`wxDrawnShape::DrawEllipticArc`](#)
[`wxDrawnShape::DrawLine`](#)
[`wxDrawnShape::DrawLines`](#)
[`wxDrawnShape::DrawPoint`](#)
[`wxDrawnShape::DrawPolygon`](#)
[`wxDrawnShape::DrawRectangle`](#)
[`wxDrawnShape::DrawRoundedRectangle`](#)
[`wxDrawnShape::DrawSpline`](#)
[`wxDrawnShape::DrawText`](#)
[`wxDrawnShape::GetAngle`](#)
[`wxDrawnShape::GetMetaFile`](#)
[`wxDrawnShape::GetRotation`](#)
[`wxDrawnShape::LoadFromMetaFile`](#)
[`wxDrawnShape::Rotate`](#)
[`wxDrawnShape::SetClippingRect`](#)
[`wxDrawnShape::SetDrawnBackgroundColour`](#)
[`wxDrawnShape::SetDrawnBackgroundMode`](#)
[`wxDrawnShape::SetDrawnBrush`](#)
[`wxDrawnShape::SetDrawnFont`](#)
[`wxDrawnShape::SetDrawnPen`](#)
[`wxDrawnShape::SetDrawnTextColour`](#)

`wxDrawnShape`

`wxdrawnshape`

`rowse00060`

`wxDrawnShape`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId('ogl.hlp', `classref')")`

wxDrawnShape::Scale

wxDrawnShape::SetSaveToFile

wxDrawnShape::Translate

`wxCircleShape`

An `wxEllipseShape` whose width and height are the same.

Derived from

`wxEllipseShape`.

Members

`wxCircleShape::wxCircleShape`

`wxCircleShape::~~wxCircleShape`

`wxCircleShape`

`wxcircleshape`

`rowse00091`

`wxCircleShape`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `classref')")`

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](#)

See also

[wxCompositeShape overview](#)

Members

[wxCompositeShape::wxCompositeShape](#)
[wxCompositeShape::~~wxCompositeShape](#)
[wxCompositeShape::AddChild](#)
[wxCompositeShape::AddConstraint](#)
[wxCompositeShape::CalculateSize](#)
[wxCompositeShape::ContainsDivision](#)
[wxCompositeShape::DeleteConstraint](#)
[wxCompositeShape::DeleteConstraintsInvolvingChild](#)
[wxCompositeShape::FindConstraint](#)
[wxCompositeShape::FindContainerImage](#)
[wxCompositeShape::GetConstraints](#)
[wxCompositeShape::GetDivisions](#)
[wxCompositeShape::MakeContainer](#)
[wxCompositeShape::OnCreateDivision](#)
[wxCompositeShape::Recompute](#)
[wxCompositeShape::RemoveChild](#)

^wxCompositeShape

^wxcompositeshape

^browse00094

^K wxCompositeShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `classref')")

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](#)

See also

[wxDividedShape overview](#)

Members

[wxDividedShape::wxDividedShape](#)
[wxDividedShape::~~wxDividedShape](#)
[wxDividedShape::EditRegions](#)
[wxDividedShape::SetRegionSizes](#)

`wxDividedShape`

`wxdividedshape`

`rowse00111`

`wxDividedShape`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId('ogl.hlp', `classref')")`

`$#+K!` 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](#)

See also

[wxCompositeShape overview](#)

Members

[wxDivisionShape::wxDivisionShape](#)
[wxDivisionShape::~~wxDivisionShape](#)
[wxDivisionShape::AdjustBottom](#)
[wxDivisionShape::AdjustLeft](#)
[wxDivisionShape::AdjustRight](#)
[wxDivisionShape::AdjustTop](#)
[wxDivisionShape::Divide](#)
[wxDivisionShape::EditEdge](#)
[wxDivisionShape::GetBottomSide](#)
[wxDivisionShape::GetHandleSide](#)
[wxDivisionShape::GetLeftSide](#)
[wxDivisionShape::GetLeftSideColour](#)
[wxDivisionShape::GetLeftSidePen](#)
[wxDivisionShape::GetRightSide](#)
[wxDivisionShape::GetTopSide](#)
[wxDivisionShape::GetTopSideColour](#)
[wxDivisionShape::GetTopSidePen](#)
[wxDivisionShape::ResizeAdjoining](#)
[wxDivisionShape::PopupMenu](#)
[wxDivisionShape::SetBottomSide](#)
[wxDivisionShape::SetHandleSide](#)
[wxDivisionShape::SetLeftSide](#)
[wxDivisionShape::SetLeftSideColour](#)
[wxDivisionShape::SetLeftSidePen](#)
[wxDivisionShape::SetRightSide](#)
[wxDivisionShape::SetTopSide](#)
[wxDivisionShape::SetTopSideColour](#)
[wxDivisionShape::SetTopSidePen](#)

`w`xDivisionShape

`w`xdivisionshape

`b`rowse00116

`K` wxDivisionShape

`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `classref')")

\$#+K! **wxEllipseShape**

The wxEllipseShape behaves similarly to the wxRectangleShape but is elliptical.

Derived from

wxShape

Members

wxEllipseShape::wxEllipseShape

wxEllipseShape::~~wxEllipseShape

^wxEllipseShape

^wxellipseshape

^browse00145

^K wxEllipseShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `classref')")

`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](#)

Members

[wxLineShape::wxLineShape](#)
[wxLineShape::~~wxLineShape](#)
[wxLineShape::AddArrow](#)
[wxLineShape::AddArrowOrdered](#)
[wxLineShape::ClearArrow](#)
[wxLineShape::ClearArrowsAtPosition](#)
[wxLineShape::DrawArrow](#)
[wxLineShape::DeleteArrowHead](#)
[wxLineShape::DeleteLineControlPoint](#)
[wxLineShape::DrawArrows](#)
[wxLineShape::DrawRegion](#)
[wxLineShape::EraseRegion](#)
[wxLineShape::FindArrowHead](#)
[wxLineShape::FindLineEndPoints](#)
[wxLineShape::FindLinePosition](#)
[wxLineShape::FindMinimumWidth](#)
[wxLineShape::FindNth](#)
[wxLineShape::GetAttachmentFrom](#)
[wxLineShape::GetAttachmentTo](#)
[wxLineShape::GetEnds](#)
[wxLineShape::GetFrom](#)
[wxLineShape::GetLabelPosition](#)
[wxLineShape::GetNextControlPoint](#)
[wxLineShape::GetTo](#)
[wxLineShape::Initialise](#)
[wxLineShape::InsertLineControlPoint](#)
[wxLineShape::IsEnd](#)
[wxLineShape::IsSpline](#)
[wxLineShape::MakeLineControlPoints](#)
[wxLineShape::OnMoveLink](#)
[wxLineShape::SetAttachmentFrom](#)

^wxLineShape

^wxlineshape

^browse00148

^K wxLineShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('ogl.hlp', `classref')")

wxLineShape::SetAttachments
wxLineShape::SetAttachmentTo
wxLineShape::SetEnds
wxLineShape::SetFrom
wxLineShape::SetIgnoreOffsets
wxLineShape::SetSpline
wxLineShape::SetTo
wxLineShape::Straighten
wxLineShape::Unlink

`$#+K!` **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](#)

Members

[wxPolygonShape::wxPolygonShape](#)
[wxPolygonShape::~~wxPolygonShape](#)
[wxPolygonShape::Create](#)
[wxPolygonShape::AddPolygonPoint](#)
[wxPolygonShape::CalculatePolygonCentre](#)
[wxPolygonShape::DeletePolygonPoint](#)
[wxPolygonShape::GetPoints](#)
[wxPolygonShape::UpdateOriginalPoints](#)

^wxPolygonShape

^wxpolygonshape

^browse00189

^K wxPolygonShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `classref')")

`$$$KL` **wxRectangleShape**

The wxRectangleShape has rounded or square corners.

Derived from

wxShape

Members

wxRectangleShape::wxRectangleShape

wxRectangleShape::~~wxRectangleShape

wxRectangleShape::SetCornerRadius

`w`xRectangleShape

`w`xrectangleshape

`b`rowse00198

`K` wxRectangleShape

`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `classref')")

`$#+K!` **wxPseudoMetaFile**

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

Derived from

wxObject

`wxPseudoMetaFile`

`wxpseudometafile`

`browse00202`

`K wxPseudoMetaFile`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `classref')")`

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](#)

Members

[wxShape::wxShape](#)
[wxShape::~~wxShape](#)
[wxShape::AddLine](#)
[wxShape::AddRegion](#)
[wxShape::AddText](#)
[wxShape::AddToCanvas](#)
[wxShape::AncestorSelected](#)
[wxShape::ApplyAttachmentOrdering](#)
[wxShape::AssignNewIds](#)
[wxShape::Attach](#)
[wxShape::AttachmentIs Valid](#)
[wxShape::AttachmentSortTest](#)
[wxShape::CalcSimpleAttachment](#)
[wxShape::CalculateSize](#)
[wxShape::ClearAttachments](#)
[wxShape::ClearRegions](#)
[wxShape::ClearText](#)
[wxShape::Constrain](#)
[wxShape::Copy](#)
[wxShape::CreateNewCopy](#)
[wxShape::DeleteControlPoints](#)
[wxShape::Detach](#)
[wxShape::Draggable](#)
[wxShape::Draw](#)
[wxShape::DrawContents](#)
[wxShape::DrawLinks](#)
[wxShape::Erase](#)
[wxShape::EraseContents](#)
[wxShape::EraseLinks](#)
[wxShape::FindRegion](#)

^wxShape

^wxshape

^browse00203

^K wxShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('ogl.hlp', `classref')")

wxShape::FindRegionNames
wxShape::Flash
wxShape::FormatText
wxShape::GetAttachmentMode
wxShape::GetAttachmentPosition
wxShape::GetBoundingBoxMax
wxShape::GetBoundingBoxMin
wxShape::GetBrush
wxShape::GetCanvas
wxShape::GetCentreResize
wxShape::GetChildren
wxShape::GetClientData
wxShape::GetDisableLabel
wxShape::GetEventHandler
wxShape::GetFixedHeight
wxShape::GetFixedSize
wxShape::GetFixedWidth
wxShape::GetFont
wxShape::GetFunctor
wxShape::GetId
wxShape::GetLinePosition
wxShape::GetLines
wxShape::GetMaintainAspectRatio
wxShape::GetNumberOfAttachments
wxShape::GetNumberOfTextRegions
wxShape::GetParent
wxShape::GetPen
wxShape::GetPerimeterPoint
wxShape::GetRegionId
wxShape::GetRegionName
wxShape::GetRegions
wxShape::GetRotation
wxShape::GetSensitivityFilter
wxShape::GetShadowMode
wxShape::GetSpaceAttachments
wxShape::GetTextColour
wxShape::GetTopAncestor
wxShape::GetX
wxShape::GetY
wxShape::HitTest
wxShape::Insert
wxShape::IsHighlighted
wxShape::IsShown
wxShape::MakeControlPoints
wxShape::MakeMandatoryControlPoints
wxShape::Move
wxShape::MoveLineToNewAttachment
wxShape::MoveLinks
wxShape::NameRegions
wxShape::Rotate
wxShape::ReadConstraints

wxShape::ReadAttributes
wxShape::ReadRegions
wxShape::Recentre
wxShape::RemoveFromCanvas
wxShape::ResetControlPoints
wxShape::ResetMandatoryControlPoints
wxShape::Recompute
wxShape::RemoveLine
wxShape::Select
wxShape::Selected
wxShape::SetAttachmentMode
wxShape::SetBrush
wxShape::SetCanvas
wxShape::SetCentreResize
wxShape::SetClientData
wxShape::SetDefaultRegionSize
wxShape::SetDisableLabel
wxShape::SetDraggable
wxShape::SetDrawHandles
wxShape::SetEventHandler
wxShape::SetFixedSize
wxShape::SetFont
wxShape::SetFormatMode
wxShape::SetHighlight
wxShape::SetId
wxShape::SetMaintainAspectRatio
wxShape::SetPen
wxShape::SetRegionName
wxShape::SetSensitivityFilter
wxShape::SetShadowMode
wxShape::SetSize
wxShape::SetSpaceAttachments
wxShape::SetTextColour
wxShape::SetX
wxShape::SetX
wxShape::SpaceAttachments
wxShape::Show
wxShape::Unlink
wxShape::WriteAttributes
wxShape::WriteRegions

wxShapeCanvas

A canvas for drawing diagrams on.

Derived from

wxScrolledWindow

See also

[wxDiagram](#)

Members

[wxShapeCanvas::wxShapeCanvas](#)
[wxShapeCanvas::~~wxShapeCanvas](#)
[wxShapeCanvas::AddShape](#)
[wxShapeCanvas::FindShape](#)
[wxShapeCanvas::FindFirstSensitiveShape](#)
[wxShapeCanvas::GetDiagram](#)
[wxShapeCanvas::GetGridSpacing](#)
[wxShapeCanvas::GetMouseTolerance](#)
[wxShapeCanvas::GetShapeList](#)
[wxShapeCanvas::GetQuickEditMode](#)
[wxShapeCanvas::InsertShape](#)
[wxShapeCanvas::OnBeginDragLeft](#)
[wxShapeCanvas::OnBeginDragRight](#)
[wxShapeCanvas::OnEndDragLeft](#)
[wxShapeCanvas::OnEndDragRight](#)
[wxShapeCanvas::OnDragLeft](#)
[wxShapeCanvas::OnDragRight](#)
[wxShapeCanvas::OnLeftClick](#)
[wxShapeCanvas::OnRightClick](#)
[wxShapeCanvas::Redraw](#)
[wxShapeCanvas::RemoveShape](#)
[wxShapeCanvas::SetDiagram](#)
[wxShapeCanvas::Snap](#)

^wxShapeCanvas

^wxshapecanvas

^browse00325

^K wxShapeCanvas

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('ogl.hlp', `classref')")

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

Members

[wxShapeEvtHandler::m_handlerShape](#)
[wxShapeEvtHandler::m_previousHandler](#)
[wxShapeEvtHandler::wxShapeEvtHandler](#)
[wxShapeEvtHandler::~wxShapeEvtHandler](#)
[wxShapeEvtHandler::CopyData](#)
[wxShapeEvtHandler::CreateNewCopy](#)
[wxShapeEvtHandler::GetPreviousHandler](#)
[wxShapeEvtHandler::GetShape](#)
[wxShapeEvtHandler::OnBeginDragLeft](#)
[wxShapeEvtHandler::OnBeginDragRight](#)
[wxShapeEvtHandler::OnBeginSize](#)
[wxShapeEvtHandler::OnChangeAttachment](#)
[wxShapeEvtHandler::OnDragLeft](#)
[wxShapeEvtHandler::OnDragRight](#)
[wxShapeEvtHandler::OnDraw](#)
[wxShapeEvtHandler::OnDrawContents](#)
[wxShapeEvtHandler::OnDrawControlPoints](#)
[wxShapeEvtHandler::OnDrawOutline](#)
[wxShapeEvtHandler::OnEndDragLeft](#)
[wxShapeEvtHandler::OnEndDragRight](#)
[wxShapeEvtHandler::OnEndSize](#)
[wxShapeEvtHandler::OnErase](#)
[wxShapeEvtHandler::OnEraseContents](#)
[wxShapeEvtHandler::OnEraseControlPoints](#)
[wxShapeEvtHandler::OnHighlight](#)
[wxShapeEvtHandler::OnLeftClick](#)
[wxShapeEvtHandler::OnMoveLink](#)
[wxShapeEvtHandler::OnMoveLinks](#)
[wxShapeEvtHandler::OnMovePost](#)

^wxShapeEvtHandler

^wxshapeevthandler

^browse00349

^K wxShapeEvtHandler

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `classref')")

wxShapeEvtHandler::OnMovePre
wxShapeEvtHandler::OnRightClick
wxShapeEvtHandler::OnSize
wxShapeEvtHandler::OnSizingBeginDragLeft
wxShapeEvtHandler::OnSizingDragLeft
wxShapeEvtHandler::OnSizingEndDragLeft
wxShapeEvtHandler::SetPreviousHandler
wxShapeEvtHandler::SetShape

\$#+K! **wxTextShape**

As wxRectangleShape, but only the text is displayed.

Derived from

wxRectangleShape

Members

wxTextShape::wxTextShape

wxTextShape::~~wxTextShape

^wxTextShape

^wxtextshape

^browse00387

^K wxTextShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `classref')")

Functions

These are the OGL functions.

[::wxOGLInitialize](#)
[::wxOGLCleanUp](#)

Functions

Functions

rowse00390

Functions

enableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `classref')")

^{\$#+K!} OGL overview

wxShapeCanvas, derived from **wxCanvas**, is the drawing area for a number of wxShape 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 organise collections of shapes, providing default file input and output behaviour.

^oGL overview

^ogloverview

^browse00394

^K OGL overview

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `topic295')")

^{\$#+K!}**wxDividedShape overview**

Classes: 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.

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;
```

^wxDividedShape overview

^dividedshapeoverview

^browse00395

^K wxDividedShape overview

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('ogl.hlp', `topic295')")

```
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();
```


\$#+K! wxCompositeShape overview

Classes: [wxCompositeShape](#), [wxOGLConstraint](#)

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](#), and add constraints using [AddConstraint](#).

After children and shapes have been added, call [Recompute](#) 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 `gdcONSTRAINT_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](#). 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](#).

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.

^wxCompositeShape overview

^compositeshapeoverview

^browse00396

^K wxCompositeShape overview

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('ogl.hlp', `topic295')")

To create an initial division, call `wxCompositeShape::MakeContainer`. Make further divisions by calling `wxDivisionShape::Divide`.

wxOGLConstraint::wxOGLConstraint

wxOGLConstraint()^K

Default constructor.

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

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:

{bmc bullet.bmp} **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

{bmc bullet.bmp} **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

{bmc bullet.bmp} **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

{bmc bullet.bmp} **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

{bmc bullet.bmp} **gyCONSTRAINT_RIGHT_OF**: the X co-ordinates of the left hand vertical edges of the bounding boxes of the constrained objects will be

^wxOGLConstraint::wxOGLConstraint

^wxoglconstraintconstr

^browse00010

^K wxOGLConstraint wxOGLConstraint

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp', `wxoglconstraint')")

^K wxOGLConstraint

^K wxOGLConstraint

greater than the X co-ordinate of the right hand vertical edge of the bounding box of the constraining object

{bmc bullet.bmp} **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

{bmc bullet.bmp} **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

{bmc bullet.bmp} **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

{bmc bullet.bmp} **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

{bmc bullet.bmp} **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

{bmc bullet.bmp} **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

{bmc bullet.bmp} **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

{bmc bullet.bmp} **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

{bmc bullet.bmp} **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

{bmc bullet.bmp} **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

$\$#+K!$ wxOGLConstraint::~~wxOGLConstraint

\sim wxOGLConstraint()^K

Destructor.

^wxOGLConstraint::~~wxOGLConstraint

^topic5

^browse00011

^K wxOGLConstraint ~wxOGLConstraint

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxoglconstraint')")

^K ~wxOGLConstraint

\$#+K!**wxOGLConstraint::Equals**

bool Equals(double x, double y)^K

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

^wxOGLConstraint::Equals

^topic6

^browse00012

^K wxOGLConstraint Equals

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxoglconstraint')")

^K Equals

\$#+K!**wxOGLConstraint::Evaluate**

bool Evaluate()^K

Evaluates this constraint, returning TRUE if anything changed.

^wxOGLConstraint::Evaluate

^topic7

^browse00013

^K wxOGLConstraint Evaluate

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxoglconstraint')")

^K Evaluate

\$#+K! **wxOGLConstraint::SetSpacing**

void SetSpacing(double *x*, double *y*)^K

Sets the horizontal and vertical spacing for the constraint.

wxOGLConstraint::SetSpacing

wxoglconstraintsetspacing

browse00014

K wxOGLConstraint SetSpacing

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxoglconstraint')")

K SetSpacing

`$#+K!wxBitmapShape::wxBitmapShape`

`wxBitmapShape()`^K

Constructor.

`wxBitmapShape::wxBitmapShape`

`topic8`

`browse00016`

`K wxBitmapShape wxBitmapShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxbitmapshape')")`

`K wxBitmapShape`

`$#+K!wxBitmapShape::~~wxBitmapShape`

`~wxBitmapShape()`^K

Destructor.

`wxBitmapShape::~~wxBitmapShape`

`topic9`

`browse00017`

`KwxBitmapShape ~wxBitmapShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxbitmapshape')")`

`K~wxBitmapShape`

\$#+KK! **wxBitmapShape::GetBitmap**

wxBitmap& GetBitmap() const

Returns a reference to the bitmap associated with this shape.

wxBitmapShape::GetBitmap

topic10

browse00018

K wxBitmapShape GetBitmap

K GetBitmap

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxbitmapshape')")

\$#+KK! **wxBitmapShape::GetFilename**

wxString GetFilename() const

Returns the bitmap filename.

wxBitmapShape::GetFilename

topic11

browse00019

K wxBitmapShape GetFilename

K GetFilename

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxbitmapshape')")

wxBitmapShape::SetBitmap

void SetBitmap(const wxBitmap& *bitmap*)^K

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.

^wwxBitmapShape::SetBitmap

^topic12

^browse00020

^K wxBitmapShape SetBitmap

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxbitmapshape')")

^K SetBitmap

\$#+K! **wxBitmapShape::SetFilename**

void SetFilename(const wxString& *filename*)^K

Sets the bitmap filename.

^wxBitmapShape::SetFilename

^topic13

^browse00021

^K wxBitmapShape SetFilename

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxbitmapshape')")

^K SetFilename

`$#+K! wxDiagram::wxDiagram`

`wxDiagram()`^K

Constructor.

`wxDiagram::wxDiagram`

`topic14`

`browse00023`

`K wxDiagram wxDiagram`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")`

`K wxDiagram`

`$#+K! wxDiagram::~~wxDiagram`

`~wxDiagram()`^K

Destructor.

`wxDiagram::~~wxDiagram`

`topic15`

`browse00024`

`K wxDiagram ~wxDiagram`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")`

`K ~wxDiagram`

`$#+K!` **wxDiagram::AddShape**

void AddShape(wxShapeshape*, wxShape **addAfter* = *NULL*)**^K

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

^wxDiagram::AddShape

^topic16

^browse00025

^K wxDiagram AddShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

^K AddShape

\$#+K! **wxDiagram::Clear**

void Clear(wxDC& dc)^K

Clears the specified device context.

^wxDiagram::Clear

^topic17

^browse00026

^K wxDiagram Clear

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

^K Clear

wxDiagram::DeleteAllShapes

void DeletesAllShapes()

Removes and deletes all shapes in the diagram.

wxDiagram::DeleteAllShapes

topic18

rowse00027

wxDiagram DeleteAllShapes

enableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp`, `wxdiagram`))

DeletesAllShapes

\$#+K! **wxDiagram::DrawOutline**

void DrawOutline(wxDC& *dc*, double *x1*, double *y1*, double *x2*, double *y2*)^K

Draws an outline rectangle on the current device context.

wxDiagram::DrawOutline

topic19

browse00028

K wxDiagram DrawOutline

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

K DrawOutline

`$#+KK!` **wxDiagram::FindShape**

wxShape* FindShape(long *id*) const

Returns the shape for the given identifier.

`wxDiagram::FindShape`

`wxdiaqramfindshape`

`rowse00029`

`K wxDiagram FindShape`

`K FindShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiaqram')")`

\$#+KK!**wxDiagram::GetCanvas**

wxShapeCanvas* GetCanvas() const

Returns the shape canvas associated with this diagram.

wxDiagram::GetCanvas

topic20

browse00030

K wxDiagram GetCanvas

K GetCanvas

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp`, `wxdiagram`)")

\$#+KK! **wxDiagram::GetCount**

int GetCount() const

Returns the number of shapes in the diagram.

wxDiagram::GetCount

wxdiagramgetcount

browse00031

K wxDiagram GetCount

K GetCount

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

`$#+KK!`**wxDiagram::GetGridSpacing**

double GetGridSpacing() const

Returns the grid spacing.

`wxDiagram::GetGridSpacing`

`topic21`

`rowse00032`

`K wxDiagram GetGridSpacing`

`K GetGridSpacing`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")`

\$#+K! **wxDiagram::GetMouseTolerance**

int GetMouseTolerance()^K

Returns the tolerance within which a mouse move is ignored.

wxDiagram::GetMouseTolerance

topic22

browse00033

K wxDiagram GetMouseTolerance

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

K GetMouseTolerance

`$#+KKl wxDiagram::GetShapeList`

`wxList* GetShapeList() const`

Returns a pointer to the internal shape list.

`wxDiagram::GetShapeList`

`topic23`

`rowse00034`

`K wxDiagram GetShapeList`

`K GetShapeList`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")`

`$#+KK!` **wxDiagram::GetQuickEditMode**

bool GetQuickEditMode() const

Returns quick edit mode.

`wxDiagram::GetQuickEditMode`

`topic24`

`rowse00035`

`K wxDiagram GetQuickEditMode`

`K GetQuickEditMode`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")`

`wxDiagram::GetSnapToGrid`

`bool GetSnapToGrid() const`

Returns snap-to-grid mode.

`wxDiagram::GetSnapToGrid`

`topic25`

`rowse00036`

`wxDiagram GetSnapToGrid`

`GetSnapToGrid`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")`

`wxDiagram::InsertShape`

`void InsertShape(wxShape *shape)`^K

Inserts a shape at the front of the shape list.

`wxDiagram::InsertShape`

`topic26`

`rowse00037`

`wxDiagram InsertShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")`

`InsertShape`

\$#+K! **wxDiagram::LoadFile**

bool LoadFile(const wxString& *filename*)^K

Loads the diagram from a file.

^wxDiagram::LoadFile

^topic27

^browse00038

^K wxDiagram LoadFile

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

^K LoadFile

`wxDiagram::OnDatabaseLoad`

`void OnDatabaseLoad(wxExprDatabase& database)`^K

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

`wxDiagram::OnDatabaseLoad`

`topic28`

`rowse00039`

`wxDiagram OnDatabaseLoad`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")`

`OnDatabaseLoad`

\$#+K! **wxDiagram::OnDatabaseSave**

void OnDatabaseSave(wxExprDatabase& *database*)^K

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

wxDiagram::OnDatabaseSave

topic29

browse00040

K wxDiagram OnDatabaseSave

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

K OnDatabaseSave

^{\$#+K!}**wxDiagram::OnHeaderLoad**

bool OnHeaderLoad(wxExprDatabase& *database*, wxExpr& *expr*)^K

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::OnHeaderLoad

^topic30

^browse00041

^K wxDiagram OnHeaderLoad

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

^K OnHeaderLoad

^{\$#+K!}**wxDiagram::OnHeaderSave**

bool OnHeaderSave(wxExprDatabase& *database*, wxExpr& *expr*)^K

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::OnHeaderSave

^topic31

^browse00042

^K wxDiagram OnHeaderSave

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

^K OnHeaderSave

\$#+K! **wxDiagram::OnShapeLoad**

bool OnShapeLoad(wxExprDatabase& *database*, wxShape& *shape*, wxExpr& *expr*)^K

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::OnShapeLoad

topic32

browse00043

K wxDiagram OnShapeLoad

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

K OnShapeLoad

\$#+K! **wxDiagram::OnShapeSave**

bool OnShapeSave(wxExprDatabase& *database*, wxShape& *shape*, wxExpr& *expr*)^K

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::OnShapeSave

^topic33

^browse00044

^K wxDiagram OnShapeSave

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

^K OnShapeSave

`$#+K! wxDiagram::ReadContainerGeometry`

`void ReadContainerGeometry(wxExprDatabase& database)K`

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

`wxDiagram::ReadContainerGeometry`

`topic34`

`rowse00045`

`K wxDiagram ReadContainerGeometry`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")`

`K ReadContainerGeometry`

`wxDiagram::ReadLines`

`void ReadLines(wxExprDatabase& database)`

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

`wxDiagram::ReadLines`

`topic35`

`rowse00046`

`wxDiagram ReadLines`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")`

`ReadLines`

`wxDiagram::ReadNodes`

`void ReadNodes(wxExprDatabase& database)`

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

`wxDiagram::ReadNodes`

`topic36`

`rowse00047`

`wxDiagram ReadNodes`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp', `wxdiagram')")`

`ReadNodes`

wxDiagram::RecentreAll

void RecentreAll(wxDC& dc)

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

wxDiagram::RecentreAll

topic37

rowse00048

wxDiagram RecentreAll

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

RecentreAll

\$#+K! **wxDiagram::Redraw**

void Redraw(wxDC& dc)^K

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

^wxDiagram::Redraw

^topic38

^browse00049

^K wxDiagram Redraw

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

^K Redraw

\$#+K! **wxDiagram::RemoveAllShapes**

void RemoveAllShapes()^K

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

wxDiagram::RemoveAllShapes

topic39

browse00050

K wxDiagram RemoveAllShapes

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp`, `wxdiagram`)"

K RemoveAllShapes

\$#+K! **wxDiagram::RemoveShape**

void RemoveShape(wxShape* *shape*)^K

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

wxDiagram::RemoveShape

topic40

browse00051

K wxDiagram RemoveShape

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

K RemoveShape

`$#+K!` **wxDiagram::SaveFile**

bool SaveFile(const wxString& *filename*)^K

Saves the diagram in a file.

`wxDiagram::SaveFile`

`topic41`

`browse00052`

`K wxDiagram SaveFile`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")`

`K SaveFile`

\$#+K! **wxDiagram::SetCanvas**

void SetCanvas(wxShapeCanvas* *canvas*)^K

Sets the canvas associated with this diagram.

^wxDiagram::SetCanvas

^wxdiagramsetcanvas

^browse00053

^K wxDiagram SetCanvas

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

^K SetCanvas

\$#+K! **wxDiagram::SetGridSpacing**

void SetGridSpacing(double *spacing*)^K

Sets the grid spacing. The default is 5.

wxDiagram::SetGridSpacing

topic42

browse00054

K wxDiagram SetGridSpacing

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

K SetGridSpacing

\$#+K! **wxDiagram::SetMouseTolerance**

void SetMouseTolerance(int *tolerance*)^K

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

^wxDiagram::SetMouseTolerance

^topic43

^browse00055

^K wxDiagram SetMouseTolerance

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

^K SetMouseTolerance

\$#+K! **wxDiagram::SetQuickEditMode**

void SetQuickEditMode(**bool** *mode*)^K

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

wxDiagram::SetQuickEditMode

topic44

browse00056

K wxDiagram SetQuickEditMode

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

K SetQuickEditMode

`$#+K!wxDiagram::SetSnapToGrid`

`void SetSnapToGrid(bool snap)K`

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

`wxDiagram::SetSnapToGrid`

`topic45`

`browse00057`

`K wxDiagram SetSnapToGrid`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")`

`K SetSnapToGrid`

\$#+K! **wxDiagram::ShowAll**

void ShowAll(**bool** *show*)^K

Calls Show for each shape in the diagram.

^wxDiagram::ShowAll

^topic46

^browse00058

^K wxDiagram ShowAll

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

^K ShowAll

\$#+K! **wxDiagram::Snap**

void Snap(double *x, double *y)K

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

wxDiagram::Snap

topic47

browse00059

K wxDiagram Snap

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdiagram')")

K Snap

`wxDrawnShape::wxDrawnShape`

`wxDrawnShape()`^K

Constructor.

`wxDrawnShape::wxDrawnShape`

`topic48`

`rowse00061`

`wxDrawnShape wxDrawnShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape")")`

`wxDrawnShape`

`$#+K!wxDrawnShape::~~wxDrawnShape`

`~wxDrawnShape()`^K

Destructor.

`wxDrawnShape::~~wxDrawnShape`

`topic49`

`browse00062`

`KwxDrawnShape ~wxDrawnShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")`

`K~wxDrawnShape`

wxDrawnShape::CalculateSize

void CalculateSize()

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

wxDrawnShape::CalculateSize

topic50

rowse00063

wxDrawnShape CalculateSize

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

CalculateSize

^{\$#+K!}**wxDrawnShape::DestroyClippingRect**

void DestroyClippingRect()^K

Destroys the clipping rectangle. See also wxDrawnShape::SetClippingRect.

^wxDrawnShape::DestroyClippingRect

^wxdrawnshapedestroyclippingrect

^browse00064

^K wxDrawnShape DestroyClippingRect

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

^K DestroyClippingRect

^{\$#+K!}**wxDrawnShape::DrawArc**

void DrawArc(const wxPoint& *centrePoint*, const wxPoint& *startPoint*, const wxPoint& *endPoint*)^K

Draws an arc (see wxWindows documentation for details).

^wxDrawnShape::DrawArc

^wxdrawnshapedrawarc

^browse00065

^K wxDrawnShape DrawArc

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

^K DrawArc

\$#+K! **wxDrawnShape::DrawAtAngle**

void DrawAtAngle(int *angle*)^K

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

{bmc bullet.bmp} ogIDRAWN_ANGLE_0

{bmc bullet.bmp} ogIDRAWN_ANGLE_90

{bmc bullet.bmp} ogIDRAWN_ANGLE_180

{bmc bullet.bmp} ogIDRAWN_ANGLE_270

See also [wxDrawnShape::GetAngle](#).

^wwxDrawnShape::DrawAtAngle

^wwxdrawnshapedrawatangle

^browse00066

^K wxDrawnShape DrawAtAngle

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

^K DrawAtAngle

\$#+K! **wxDrawnShape::DrawEllipticArc**

void DrawEllipticArc(const wxRect& *rect*, double *startAngle*, double *endAngle*)^K

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

wxDrawnShape::DrawEllipticArc

wxdrawnshapedrawellipticarc

browse00067

K wxDrawnShape DrawEllipticArc

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

K DrawEllipticArc

\$#+K! **wxDrawnShape::DrawLine**

void DrawLine(const wxPoint& *point1*, const wxPoint& *point2*)^K

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

wxDrawnShape::DrawLine

wxdrawnshapedrawline

browse00068

K wxDrawnShape DrawLine

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

K DrawLine

\$#+K! **wxDrawnShape::DrawLines**

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

Draws *n* lines.

wxDrawnShape::DrawLines

wxdrawnshapedrawlines

browse00069

K wxDrawnShape DrawLines

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

K DrawLines

\$#+K! **wxDrawnShape::DrawPoint**

void DrawPoint(const wxPoint& *point*)^K

Draws a point.

wxDrawnShape::DrawPoint

wxdrawnshapedrawpoint

browse00070

K wxDrawnShape DrawPoint

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

K DrawPoint

\$#+K! **wxDrawnShape::DrawPolygon**

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

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::DrawPolygon

wxdrawnshapedrawpolygon

browse00071

K wxDrawnShape DrawPolygon

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

K DrawPolygon

\$#+K! **wxDrawnShape::DrawRectangle**

void DrawRectangle(const wxRect& *rect*)^K

Draws a rectangle.

^wxDrawnShape::DrawRectangle

^wxdrawnshapedrawrectangle

^browse00072

^K wxDrawnShape DrawRectangle

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

^K DrawRectangle

\$#+K! **wxDrawnShape::DrawRoundedRectangle**

void DrawRoundedRectangle(const wxRect& *rect*, double *radius*)^K

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::DrawRoundedRectangle

wxdrawnshapedrawroundedrectangle

browse00073

K wxDrawnShape DrawRoundedRectangle

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape")")

K DrawRoundedRectangle

wxDrawnShape::DrawSpline

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

Draws a spline curve.

^wxDrawnShape::DrawSpline

^wxdrawnshapedrawspline

^browse00074

^K wxDrawnShape DrawSpline

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

^K DrawSpline

^{\$#+K!}**wxDrawnShape::DrawText**

void DrawText(const wxString& *text*, const wxPoint& *point*)^K

Draws text at the given point.

^wxDrawnShape::DrawText

^wxdrawnshapedrawtext

^browse00075

^K wxDrawnShape DrawText

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape)")

^K DrawText

\$#+KK! **wxDrawnShape::GetAngle**

int GetAngle() const

Returns the current orientation, which can be one of:

{bmc bullet.bmp} ogIDRAWN_ANGLE_0

{bmc bullet.bmp} ogIDRAWN_ANGLE_90

{bmc bullet.bmp} ogIDRAWN_ANGLE_180

{bmc bullet.bmp} ogIDRAWN_ANGLE_270

See also [wxDrawnShape::DrawAtAngle](#).

^wxDrawnShape::GetAngle

^wxdrawnshapegetangle

^rowse00076

^wxDrawnShape GetAngle

^GetAngle

^nableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape")")

`$#+KK!` **wxDrawnShape::GetMetaFile**

wxPseudoMetaFile& GetMetaFile() const

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

`wxDrawnShape::GetMetaFile`

`topic51`

`rowse00077`

`K wxDrawnShape GetMetaFile`

`K GetMetaFile`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape")")`

`wxDrawnShape::GetRotation`

`double GetRotation() const`

Returns the current rotation of the shape in radians.

`wxDrawnShape::GetRotation`

`wxdrawnshapegetrotation`

`rowse00078`

`wxDrawnShape GetRotation`

`GetRotation`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")`

\$#+K! **wxDrawnShape::LoadFromMetaFile**

bool LoadFromMetaFile(const wxString& *filename*)^K

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

wxDrawnShape::LoadFromMetaFile

topic52

browse00079

K wxDrawnShape LoadFromMetaFile

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

K LoadFromMetaFile

\$#+K! **wxDrawnShape::Rotate**

void Rotate(double *x*, double *y*, double *theta*)^K

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

wxDrawnShape::Rotate

topic53

browse00080

K wxDrawnShape Rotate

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape")")

K Rotate

^{\$#+K!}**wxDrawnShape::SetClippingRect**

void SetClippingRect(const wxRect& *rect*)^K

Sets the clipping rectangle. See also [wxDrawnShape::DestroyClippingRect](#).

^wxDrawnShape::SetClippingRect

^wxdrawnshapesetclippingrect

^browse00081

^K wxDrawnShape SetClippingRect

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

^K SetClippingRect

^{\$#+K!}**wxDrawnShape::SetDrawnBackgroundColour**

void SetDrawnBackgroundColour(const wxColour& *colour*)^K

Sets the current background colour for the current metafile.

^wxDrawnShape::SetDrawnBackgroundColour

^wxdrawnshapesetdrawnbackgroundcolour

^browse00082

^K wxDrawnShape SetDrawnBackgroundColour

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape)")

^K SetDrawnBackgroundColour

\$#+K! **wxDrawnShape::SetDrawnBackgroundMode**

void SetDrawnBackgroundMode(int *mode*)^K

Sets the current background mode for the current metafile.

^wxDrawnShape::SetDrawnBackgroundMode

^wxdrawnshapesetdrawnbackgroundmode

^browse00083

^K wxDrawnShape SetDrawnBackgroundMode

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape)")

^K SetDrawnBackgroundMode

\$#+K! **wxDrawnShape::SetDrawnBrush**

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

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::SetDrawnBrush

Wxdrawnshapesetdrawnbrush

browse00084

K wxDrawnShape SetDrawnBrush

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

K SetDrawnBrush

\$#+K! **wxDrawnShape::SetDrawnFont**

void SetDrawnFont(**wxFont*** *font*)^K

Sets the current font for the current metafile.

wxDrawnShape::SetDrawnFont

wxdrawnshapesetdrawnfont

browse00085

K wxDrawnShape SetDrawnFont

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

K SetDrawnFont

\$#+K! **wxDrawnShape::SetDrawnPen**

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

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::SetDrawnPen

wxdrawnshapesetdrawnpen

browse00086

K wxDrawnShape SetDrawnPen

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape)")

K SetDrawnPen

^{\$#+K!}**wxDrawnShape::SetDrawnTextColour**

void SetDrawnTextColour(const wxColour& *colour*)^K

Sets the current text colour for the current metafile.

^wxDrawnShape::SetDrawnTextColour

^wxdrawnshapesetdrawntextcolour

^browse00087

^K wxDrawnShape SetDrawnTextColour

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape)")

^K SetDrawnTextColour

\$#+K! **wxDrawnShape::Scale**

void Scale(double sx, double sy)^K

Scales the shape by the given amount.

wxDrawnShape::Scale

topic54

browse00088

K wxDrawnShape Scale

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape)")

K Scale

\$#+K! **wxDrawnShape::SetSaveToFile**

void SetSaveToFile(**bool** *save*)^K

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::SetSaveToFile

topic55

browse00089

K wxDrawnShape SetSaveToFile

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape')")

K SetSaveToFile

\$#+K! **wxDrawnShape::Translate**

void Translate(double x, double y)^K

Translates the shape by the given amount.

wxDrawnShape::Translate

topic56

browse00090

K wxDrawnShape Translate

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdrawnshape")")

K Translate

`$#+K!wxCircleShape::wxCircleShape`

`wxCircleShape(double width = 0.0)K`

Constructor.

`wxCircleShape::wxCircleShape`

`topic57`

`browse00092`

`K wxCircleShape wxCircleShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxcirclesshape')")`

`K wxCircleShape`

`$#+K!wxCircleShape::~~wxCircleShape`

`~wxCircleShape()`^K

Destructor.

`wxCircleShape::~~wxCircleShape`

`topic58`

`browse00093`

`KwxCircleShape ~wxCircleShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxcircleshape')")`

`K ~wxCircleShape`

`wxCompositeShape::wxCompositeShape`

`wxCompositeShape()`^K

Constructor.

`wxCompositeShape::wxCompositeShape`

`topic59`

`browse00095`

`wxCompositeShape wxCompositeShape`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',`

``wxcompositeshape')")`

`wxCompositeShape`

\$#+K! **wxCompositeShape::~wxCompositeShape**

~wxCompositeShape()^K

Destructor.

wxCompositeShape::~wxCompositeShape

topic60

browse00096

K wxCompositeShape ~wxCompositeShape

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',

`wxcompositeshape')

K ~wxCompositeShape

\$#+K! **wxCompositeShape::AddChild**

void AddChild(wxShape *child, wxShape *addAfter = NULL)^K

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

wxCompositeShape::AddChild

wxcompositeshapeaddchild

browse00097

K wxCompositeShape AddChild

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxcompositeshape')")

K AddChild

`wxCompositeShape::AddConstraint`

`wxOGLConstraint * AddConstraint(wxOGLConstraint *constraint)`^K

`wxOGLConstraint * AddConstraint(int type, wxShape *constraining, wxList&constrained)`^K

`wxOGLConstraint * AddConstraint(int type, wxShape *constraining, wxShape *constrained)`^K

Adds a constraint to the composite.

^wxCompositeShape::AddConstraint

^wxcompositeshapeaddconstraint

^browse00098

^K wxCompositeShape AddConstraint

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('ogl.hlp',

`wxcompositeshape'))

^K AddConstraint

^K AddConstraint

^K AddConstraint

wxCompositeShape::CalculateSize

void CalculateSize()

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

wxCompositeShape::CalculateSize

topic61

rowse00099

wxCompositeShape CalculateSize

**enableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxcompositeshape')")**

CalculateSize

\$\$\$+K! **wxCompositeShape::ContainsDivision**

bool FindContainerImage(wxDivisionShape *division)^K

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

wxCompositeShape::ContainsDivision

topic62

browse00100

K wxCompositeShape ContainsDivision

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',

`wxcompositeshape')

K FindContainerImage

wxCompositeShape::DeleteConstraint

void DeleteConstraint(wxOGLConstraint *constraint)^K

Deletes constraint from composite.

^wxCompositeShape::DeleteConstraint

^topic63

^browse00101

^K wxCompositeShape DeleteConstraint

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxcompositeshape')")

^K DeleteConstraint

`wxCompositeShape::DeleteConstraintsInvolvingChild`

`void DeleteConstraintsInvolvingChild(wxShape *child)`^K

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

`wxCompositeShape::DeleteConstraintsInvolvingChild`

`topic64`

`rowse00102`

`wxCompositeShape DeleteConstraintsInvolvingChild`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp',
`wxcompositeshape')")`

`DeleteConstraintsInvolvingChild`

wxCompositeShape::FindConstraint

wxOGLConstraint * FindConstraint(long id, wxCompositeShape **actualComposite)^K

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::FindConstraint

^topic65

^browse00103

^K wxCompositeShape FindConstraint

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxcompositeshape')")

^K FindConstraint

wxCompositeShape::FindContainerImage

wxShape * FindContainerImage()^K

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

^wxCompositeShape::FindContainerImage

^topic66

^browse00104

^K wxCompositeShape FindContainerImage

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',

[`]wxcompositeshape')

^K FindContainerImage

wxCompositeShape::GetConstraints

wxList& GetConstraints() const

Returns a reference to the list of constraints.

wxCompositeShape::GetConstraints

topic67

browse00105

wxCompositeShape GetConstraints

GetConstraints

**EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp',
`wxcompositeshape`"))**

`$#+KK!` **wxCompositeShape::GetDivisions**

wxList& GetDivisions() const

Returns a reference to the list of divisions.

`w`xCompositeShape::GetDivisions

`t`opic68

`b`rowse00106

`K` wxCompositeShape GetDivisions

`K` GetDivisions

`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp',
`wxcompositeshape`")")

`$#+K!` **wxCompositeShape::MakeContainer**

void MakeContainer()^K

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

`wxCompositeShape::MakeContainer`

`wxcompositeshapemakecontainer`

`browse00107`

`K wxCompositeShape MakeContainer`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',`

``wxcompositeshape')")`

`K MakeContainer`

wxCompositeShape::OnCreateDivision

wxDivisionShape * OnCreateDivision()^K

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

^wxCompositeShape::OnCreateDivision

^topic69

^browse00108

^K wxCompositeShape OnCreateDivision

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxcompositeshape')")

^K OnCreateDivision

`$#+K! wxCompositeShape::Recompute`

`bool Recompute()`^K

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

`w`xCompositeShape::Recompute
`w`xcompositeshaperecompute
`b`rowse00109
^K wxCompositeShape Recompute
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxcompositeshape')")
^K Recompute

\$#+K! **wxCompositeShape::RemoveChild**

void RemoveChild(wxShape *child)^K

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

wxCompositeShape::RemoveChild

topic70

browse00110

K wxCompositeShape RemoveChild

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxcompositeshape')")

K RemoveChild

`$#+K!wxDividedShape::wxDividedShape`

`wxDividedShape(double width = 0.0, double height = 0.0)K`

Constructor.

`wxDividedShape::wxDividedShape`

`topic71`

`browse00112`

`KwxDividedShape wxDividedShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdividedshape')")`

`KwxDividedShape`

`$#+K!wxDividedShape::~~wxDividedShape`

`~wxDividedShape()`^K

Destructor.

`wxDividedShape::~~wxDividedShape`

`topic72`

`browse00113`

`KwxDividedShape ~wxDividedShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdividedshape')")`

`K~wxDividedShape`

\$#+K! **wxDividedShape::EditRegions**

void EditRegions()^K

Edit the region colours and styles.

^wxDividedShape::EditRegions

^topic73

^browse00114

^K wxDividedShape EditRegions

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdividedshape')")

^K EditRegions

`$#+K!wxDividedShape::SetRegionSizes`

`void SetRegionSizes()`^K

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

`wxDividedShape::SetRegionSizes`

`topic74`

`browse00115`

`K wxDividedShape SetRegionSizes`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdividedshape')")`

`K SetRegionSizes`

`$#+K!wxDivisionShape::wxDivisionShape`

`wxDivisionShape()`^K

Constructor.

`wxDivisionShape::wxDivisionShape`

`topic75`

`browse00117`

`KwxDivisionShape wxDivisionShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")`

`KwxDivisionShape`

`$#+K!wxDivisionShape::~~wxDivisionShape`

`~wxDivisionShape()`^K

Destructor.

`wxDivisionShape::~~wxDivisionShape`

`topic76`

`browse00118`

`KwxDivisionShape ~wxDivisionShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")`

`K ~wxDivisionShape`

\$#+K! **wxDivisionShape::AdjustBottom**

void AdjustBottom(double *bottom*, bool *test*)^K

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

^wxDivisionShape::AdjustBottom

^topic77

^browse00119

^K wxDivisionShape AdjustBottom

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

^K AdjustBottom

\$#+K! **wxDivisionShape::AdjustLeft**

void AdjustLeft(double *left*, bool *test*)^K

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

^wxDivisionShape::AdjustLeft

^topic78

^browse00120

^K wxDivisionShape AdjustLeft

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

^K AdjustLeft

\$#+K! **wxDivisionShape::AdjustRight**

void AdjustRight(double *right*, bool *test*)^K

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

^wxDivisionShape::AdjustRight

^topic79

^browse00121

^K wxDivisionShape AdjustRight

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

^K AdjustRight

\$#+K! **wxDivisionShape::AdjustTop**

void AdjustTop(double *top*, bool *test*)^K

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

^wxDivisionShape::AdjustTop

^topic80

^browse00122

^K wxDivisionShape AdjustTop

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

^K AdjustTop

\$#+K! **wxDivisionShape::Divide**

void Divide(int *direction*)^K

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

wxDivisionShape::Divide

wxdivisionshapedivide

browse00123

K wxDivisionShape Divide

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

K Divide

\$#+K! **wxDivisionShape::EditEdge**

void EditEdge(int *side*)^K

Interactively edit style of left or top side.

^wxDivisionShape::EditEdge

^topic81

^browse00124

^K wxDivisionShape EditEdge

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

^K EditEdge

`wxDivisionShape::GetBottomSide`

`wxDivisionShape * GetBottomSide()`^K

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

`wxDivisionShape::GetBottomSide`

^topic82

^browse00125

^K `wxDivisionShape GetBottomSide`

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

^K `GetBottomSide`

wxDivisionShape::GetHandleSide

int GetHandleSide()^K

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

^wxDivisionShape::GetHandleSide

^topic83

^browse00126

^K wxDivisionShape GetHandleSide

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

^K GetHandleSide

`$#+K! wxDivisionShape::GetLeftSide`

`wxDivisionShape * GetLeftSide()`^K

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

`wxDivisionShape::GetLeftSide`

`topic84`

`browse00127`

`K wxDivisionShape GetLeftSide`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")`

`K GetLeftSide`

`$#+K! wxDivisionShape::GetLeftSideColour`

`wxString GetLeftSideColour()`^K

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

`wxDivisionShape::GetLeftSideColour`

`topic85`

`rowse00128`

`K wxDivisionShape GetLeftSideColour`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")`

`K GetLeftSideColour`

`$#+K! wxDivisionShape::GetLeftSidePen`

`wxPen * GetLeftSidePen()`^K

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

^wxDivisionShape::GetLeftSidePen

^topic86

^browse00129

^K wxDivisionShape GetLeftSidePen

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

^K GetLeftSidePen

`$#+K! wxDivisionShape::GetRightSide`

`wxDivisionShape * GetRightSide()`^K

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

^wxDivisionShape::GetRightSide

^topic87

^browse00130

^K wxDivisionShape GetRightSide

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp`, `wxdivisionshape`)"

^K GetRightSide

`$#+K!wxDivisionShape::GetTopSide`

`wxDivisionShape * GetTopSide()`^K

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

`wxDivisionShape::GetTopSide`

`topic88`

`rowse00131`

`K wxDivisionShape GetTopSide`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")`

`K GetTopSide`

\$#+K! **wxDivisionShape::GetTopSideColour**

wxString GetTopSideColour()^K

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

wxDivisionShape::GetTopSideColour

topic89

browse00132

K wxDivisionShape GetTopSideColour

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp`, `wxdivisionshape`)"

K GetTopSideColour

`$#+K!wxDivisionShape::GetTopSidePen`

`wxPen * GetTopSidePen()`^K

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

`wxDivisionShape::GetTopSidePen`

`topic90`

`browse00133`

`K wxDivisionShape GetTopSidePen`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp`, `wxdivisionshape`)"`

`K GetTopSidePen`

\$#+K! **wxDivisionShape::ResizeAdjoining**

void ResizeAdjoining(int *side*, double *newPos*, bool *test*)^K

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:

- {bmc bullet.bmp} DIVISION_SIDE_NONE
- {bmc bullet.bmp} DIVISION_SIDE_LEFT
- {bmc bullet.bmp} DIVISION_SIDE_TOP
- {bmc bullet.bmp} DIVISION_SIDE_RIGHT
- {bmc bullet.bmp} DIVISION_SIDE_BOTTOM

^wxDivisionShape::ResizeAdjoining

^topic91

^browse00134

^K wxDivisionShape ResizeAdjoining

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

^K ResizeAdjoining

`$#+K! wxDivisionShape::PopupMenu`

`void PopupMenu(double x, double y)K`

PopupMenu the division menu.

`wxDivisionShape::PopupMenu`

`topic92`

`browse00135`

`K wxDivisionShape PopupMenu`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")`

`K PopupMenu`

wxDivisionShape::SetBottomSide

void SetBottomSide(wxDivisionShape **shape*)^K

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

^wxDivisionShape::SetBottomSide

^topic93

^browse00136

^K wxDivisionShape SetBottomSide

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

^K SetBottomSide

\$#+K! **wxDivisionShape::SetHandleSide**

int SetHandleSide()^K

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

wxDivisionShape::SetHandleSide

topic94

browse00137

K wxDivisionShape SetHandleSide

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

K SetHandleSide

\$#+K! **wxDivisionShape::SetLeftSide**

void SetLeftSide(wxDivisionShape **shape*)^K

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

wxDivisionShape::SetLeftSide

topic95

browse00138

K wxDivisionShape SetLeftSide

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

K SetLeftSide

\$#+K! **wxDivisionShape::SetLeftSideColour**

void SetLeftSideColour(const wxString& colour)^K

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

wxDivisionShape::SetLeftSideColour

topic96

browse00139

K wxDivisionShape SetLeftSideColour

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

K SetLeftSideColour

`$#+K!wxDivisionShape::SetLeftSidePen`

`void SetLeftSidePen(wxPen *pen)K`

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

`wxDivisionShape::SetLeftSidePen`

`topic97`

`browse00140`

`K wxDivisionShape SetLeftSidePen`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")`

`K SetLeftSidePen`

\$#+K! **wxDivisionShape::SetRightSide**

void SetRightSide(wxDivisionShape **shape*)^K

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

^wxDivisionShape::SetRightSide

^topic98

^browse00141

^K wxDivisionShape SetRightSide

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

^K SetRightSide

\$#+K! **wxDivisionShape::SetTopSide**

void SetTopSide(wxDivisionShape **shape*)^K

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

^wxDivisionShape::SetTopSide

^topic99

^browse00142

^K wxDivisionShape SetTopSide

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

^K SetTopSide

\$#+K! **wxDivisionShape::SetTopSideColour**

void SetTopSideColour(const wxString& *colour*)^K

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

wxDivisionShape::SetTopSideColour

topic100

browse00143

K wxDivisionShape SetTopSideColour

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")

K SetTopSideColour

`$#+K!wxDivisionShape::SetTopSidePen`

`void SetTopSidePen(wxPen *pen)K`

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

`wxDivisionShape::SetTopSidePen`

`topic101`

`browse00144`

`K wxDivisionShape SetTopSidePen`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxdivisionshape')")`

`K SetTopSidePen`

`$#+K!wxEllipseShape::wxEllipseShape`

`wxEllipseShape(double width = 0.0, double height = 0.0)K`

Constructor.

`wxEllipseShape::wxEllipseShape`

`topic102`

`browse00146`

`K wxEllipseShape wxEllipseShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxellipseshape')")`

`K wxEllipseShape`

\$#+K! **wxEllipseShape::~~wxEllipseShape**

~wxEllipseShape()^K

Destructor.

^wxEllipseShape::~~wxEllipseShape

^topic103

^browse00147

^K wxEllipseShape ~wxEllipseShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxellipseshape')")

^K ~wxEllipseShape

`$#+K! wxLineShape::wxLineShape`

`wxLineShape()`^K

Constructor.

Usually you will call `wxLineShape::MakeLineControlPoints` to specify the number of segments in the line.

`wxLineShape::wxLineShape`

`topic104`

`browse00149`

`K wxLineShape wxLineShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")`

`K wxLineShape`

\$#+K! **wxLineShape::~~wxLineShape**

~wxLineShape()^K

Destructor.

wxLineShape::~~wxLineShape

topic105

browse00150

K wxLineShape ~wxLineShape

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

K ~wxLineShape

`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)`^K

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.

^w`wxLineShape::AddArrow`

^w`wxlineshapeaddarrow`

^b`rowse00151`

^K`wxLineShape AddArrow`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")`

^K`AddArrow`

`$#+K! wxLineShape::AddArrowOrdered`

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

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::AddArrowOrdered`

`topic106`

`browse00152`

`K wxLineShape AddArrowOrdered`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")`

`K AddArrowOrdered`

^{\$#+K!}**wxLineShape::ClearArrow**

bool ClearArrow(const wxString& *name*)^K

Delete the arrow with the given name.

^wxLineShape::ClearArrow

^topic107

^browse00153

^K wxLineShape ClearArrow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K ClearArrow

\$#+K! **wxLineShape::ClearArrowsAtPosition**

void ClearArrowsAtPosition(int *position* = -1)^K

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

^wwxLineShape::ClearArrowsAtPosition

^topic108

^browse00154

^K wxLineShape ClearArrowsAtPosition

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K ClearArrowsAtPosition

\$#+K! **wxLineShape::DrawArrow**

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

Draws the given arrowhead (or annotation).

^wxLineShape::DrawArrow

^topic109

^browse00155

^K wxLineShape DrawArrow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K DrawArrow

`$#+K! wxLineShape::DeleteArrowHead`

`bool DeleteArrowHead(long arrowId)K`

`bool DeleteArrowHead(int position, const wxString& name)K`

Delete arrowhead by id or position and name.

`wxLineShape::DeleteArrowHead`

`topic110`

`browse00156`

`K wxLineShape DeleteArrowHead`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp', `wxlineshape')")`

`K DeleteArrowHead`

`K DeleteArrowHead`

\$#+K! **wxLineShape::DeleteLineControlPoint**

bool DeleteLineControlPoint()^K

Deletes an arbitrary point on the line.

wxLineShape::DeleteLineControlPoint

topic111

browse00157

K wxLineShape DeleteLineControlPoint

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

K DeleteLineControlPoint

\$#+K! **wxLineShape::DrawArrows**

void DrawArrows(wxDC& dc)^K

Draws all arrows.

wxLineShape::DrawArrows

topic112

browse00158

K wxLineShape DrawArrows

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

K DrawArrows

wxLineShape::DrawRegion

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

Format one region at this position.

^wxLineShape::DrawRegion

^topic113

^browse00159

^K wxLineShape DrawRegion

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K DrawRegion

wxLineShape::EraseRegion

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

Format one region at this position.

^wxLineShape::EraseRegion

^topic114

^browse00160

^K wxLineShape EraseRegion

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K EraseRegion

\$#+K! **wxLineShape::FindArrowHead**

wxArrowHead * FindArrowHead(long *arrowId*)^K

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

Find arrowhead by id or position and name.

wxLineShape::FindArrowHead

topic115

browse00161

K wxLineShape FindArrowHead

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp', `wxlineshape')")

K FindArrowHead

K FindArrowHead

`$#+K! wxLineShape::FindLineEndPoints`

`void FindLineEndPoints(double *fromX, double *fromY, double *toX, double *toY)`^K

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::FindLineEndPoints

^topic116

^browse00162

^K wxLineShape FindLineEndPoints

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K FindLineEndPoints

\$#+K! **wxLineShape::FindLinePosition**

int FindLinePosition(double x, double y)^K

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

^wxLineShape::FindLinePosition

^topic117

^browse00163

^K wxLineShape FindLinePosition

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K FindLinePosition

`$#+K! wxLineShape::FindMinimumWidth`

`double FindMinimumWidth()`^K

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

`wxLineShape::FindMinimumWidth`

`topic118`

`rowse00164`

`K wxLineShape FindMinimumWidth`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")`

`K FindMinimumWidth`

^{\$#+K!}**wxLineShape::FindNth**

void FindNth(**wxShape** **image*, **int** **nth*, **int** **noArcs*, **bool** *incoming*)^K

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

^wxLineShape::FindNth

^topic119

^browse00165

^K wxLineShape FindNth

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K FindNth

`$#+KKl wxLineShape::GetAttachmentFrom`

`int GetAttachmentFrom() const`

Returns the attachment point on the 'from' node.

`wxLineShape::GetAttachmentFrom`

`topic120`

`rowse00166`

`K wxLineShape GetAttachmentFrom`

`K GetAttachmentFrom`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")`

`$#+KK!wxLineShape::GetAttachmentTo`

`int GetAttachmentTo() const`

Returns the attachment point on the 'to' node.

`wxLineShape::GetAttachmentTo`

`topic121`

`rowse00167`

`K wxLineShape GetAttachmentTo`

`K GetAttachmentTo`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")`

\$#+K! **wxLineShape::GetEnds**

void GetEnds(double *x1, double *y1, double *x2, double *y2)^K

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

^wxLineShape::GetEnds

^topic122

^browse00168

^K wxLineShape GetEnds

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K GetEnds

`$#+KKl wxLineShape::GetFrom`

`wxShape * GetFrom() const`

Gets the 'from' object.

`wxLineShape::GetFrom`

`topic123`

`browse00169`

`K wxLineShape GetFrom`

`K GetFrom`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")`

`wxLineShape::GetLabelPosition`

`void GetLabelPosition(int position, double *x, double *y)`^K

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

`wxLineShape::GetLabelPosition`

`topic124`

`rowse00170`

`wxLineShape GetLabelPosition`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp', `wxlineshape')")`

`GetLabelPosition`

^{\$#+K!}**wxLineShape::GetNextControlPoint**

wxPoint * GetNextControlPoint(wxShape *shape)^K

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::GetNextControlPoint

^topic125

^browse00171

^K wxLineShape GetNextControlPoint

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K GetNextControlPoint

wxLineShape::GetTo

wxShape * GetTo()

Gets the 'to' object.

wxLineShape::GetTo

topic126

browse00172

wxLineShape GetTo

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp', `wxlineshape')")

GetTo

`$#+K!wxLineShape::Initialise`

`void Initialise()`^K

Initialises the line object.

`wxLineShape::Initialise`

`topic127`

`rowse00173`

`K wxLineShape Initialise`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")`

`K Initialise`

`wxLineShape::InsertLineControlPoint`

`void InsertLineControlPoint()`^K

Inserts a control point at an arbitrary position.

`wxLineShape::InsertLineControlPoint`

^topic128

^browse00174

^K `wxLineShape InsertLineControlPoint`

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K `InsertLineControlPoint`

\$#+K! **wxLineShape::IsEnd**

bool IsEnd(wxShape **shape*)^K

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

wxLineShape::IsEnd

topic129

browse00175

K wxLineShape IsEnd

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

K IsEnd

\$#+K! **wxLineShape::IsSpline**

bool IsSpline()^K

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

wxLineShape::IsSpline

topic130

browse00176

K wxLineShape IsSpline

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

K IsSpline

wxLineShape::MakeLineControlPoints

void MakeLineControlPoints(int *n*)^K

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

^wxLineShape::MakeLineControlPoints

^wxlineshapemakelinecontrolpoints

^browse00177

^K wxLineShape MakeLineControlPoints

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K MakeLineControlPoints

\$#+K! **wxLineShape::OnMoveLink**

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

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

wxLineShape::OnMoveLink

topic131

browse00178

K wxLineShape OnMoveLink

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

K OnMoveLink

^{\$#+K!}**wxLineShape::SetAttachmentFrom**

void SetAttachmentTo(int *fromAttach*)^K

Sets the 'from' shape attachment.

^wxLineShape::SetAttachmentFrom

^topic132

^browse00179

^K wxLineShape SetAttachmentFrom

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K SetAttachmentTo

\$#+K! **wxLineShape::SetAttachments**

void SetAttachments(int *fromAttach*, int *toAttach*)^K

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

^wxLineShape::SetAttachments

^topic133

^browse00180

^K wxLineShape SetAttachments

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K SetAttachments

\$#+K! **wxLineShape::SetAttachmentTo**

void SetAttachmentTo(int *toAttach*)^K

Sets the 'to' shape attachment.

^wxLineShape::SetAttachmentTo

^topic134

^browse00181

^K wxLineShape SetAttachmentTo

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K SetAttachmentTo

\$#+K! **wxLineShape::SetEnds**

void SetEnds(double x1, double y1, double x2, double y2)^K

Sets the end positions of the line.

wxLineShape::SetEnds

topic135

browse00182

K wxLineShape SetEnds

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

K SetEnds

\$#+K! **wxLineShape::SetFrom**

void SetFrom(wxShape *object)^K

Sets the 'from' object for the line.

^wxLineShape::SetFrom

^topic136

^browse00183

^K wxLineShape SetFrom

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K SetFrom

\$#+K! **wxLineShape::SetIgnoreOffsets**

void SetIgnoreOffsets(**bool** *ignore*)^K

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

wxLineShape::SetIgnoreOffsets

topic137

browse00184

K wxLineShape SetIgnoreOffsets

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

K SetIgnoreOffsets

\$#+K! **wxLineShape::SetSpline**

void SetSpline(**bool** *spline*)^K

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

wxLineShape::SetSpline

topic138

browse00185

K wxLineShape SetSpline

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

K SetSpline

`$#+K!wxLineShape::SetTo`

`void SetTo(wxShape *object)K`

Sets the 'to' object for the line.

`wxLineShape::SetTo`

`topic139`

`browse00186`

`K wxLineShape SetTo`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")`

`K SetTo`

`wxLineShape::Straighten`

`void Straighten(wxDC* dc = NULL)`

Straighten verticals and horizontals. *dc* is optional.

`wxLineShape::Straighten`

`topic140`

`rowse00187`

`wxLineShape Straighten`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp', `wxlineshape')")`

`Straighten`

\$#+K! **wxLineShape::Unlink**

void Unlink()^K

Unlinks the line from the nodes at either end.

^wxLineShape::Unlink

^topic141

^browse00188

^K wxLineShape Unlink

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxlineshape')")

^K Unlink

`wxPolygonShape::wxPolygonShape`

`wxPolygonShape(void)`^K

Constructor. Call `wxPolygonShape::Create` to specify the polygon's vertices.

`wxPolygonShape::wxPolygonShape`

`topic142`

`browse00190`

`wxPolygonShape wxPolygonShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxpolygonshape')")`

`wxPolygonShape`

`$#+K! wxPolygonShape::~~wxPolygonShape`

`~wxPolygonShape()`^K

Destructor.

`wxPolygonShape::~~wxPolygonShape`

`topic143`

`browse00191`

`K wxPolygonShape ~wxPolygonShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxpolygonshape')")`

`K ~wxPolygonShape`

\$#+K! **wxPolygonShape::Create**

void Create(wxList* *points*)^K

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::Create

wxpolygonshapecreate

browse00192

K wxPolygonShape Create

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxpolygonshape')")

K Create

wxPolygonShape::AddPolygonPoint

void AddPolygonPoint(int *pos* = 0)^K

Add a control point after the given point.

^wxPolygonShape::AddPolygonPoint

^topic144

^browse00193

^K wxPolygonShape AddPolygonPoint

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxpolygonshape')")

^K AddPolygonPoint

`$#+K! wxPolygonShape::CalculatePolygonCentre`

`void CalculatePolygonCentre()`^K

Recalculates the centre of the polygon.

^wxPolygonShape::CalculatePolygonCentre

^topic145

^browse00194

^K wxPolygonShape CalculatePolygonCentre

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxpolygonshape')")

^K CalculatePolygonCentre

`wxPolygonShape::DeletePolygonPoint`

`void DeletePolygonPoint(int pos = 0)`^K

Deletes a control point.

`wxPolygonShape::DeletePolygonPoint`

^topic146

^browse00195

^K `wxPolygonShape DeletePolygonPoint`

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxpolygonshape')")

^K `DeletePolygonPoint`

wxPolygonShape::GetPoints

wxList * GetPoints()

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

wxPolygonShape::GetPoints

topic147

rowse00196

wxPolygonShape GetPoints

enableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxpolygonshape')")

GetPoints

^{\$#+K!}**wxPolygonShape::UpdateOriginalPoints**

void UpdateOriginalPoints()^K

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

^wxPolygonShape::UpdateOriginalPoints

^topic148

^browse00197

^K wxPolygonShape UpdateOriginalPoints

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp`, `wxpolygonshape`)"

^K UpdateOriginalPoints

`wxRectangleShape::wxRectangleShape`

`wxRectangleShape(double width = 0.0, double height = 0.0)`^K

Constructor.

`wxRectangleShape::wxRectangleShape`

`topic149`

`browse00199`

`wxRectangleShape wxRectangleShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxrectangleshape'))`

`wxRectangleShape`

`wxRectangleShape::~wxRectangleShape`

`~wxRectangleShape()`^K

Destructor.

`wxRectangleShape::~wxRectangleShape`

^topic150

^browse00200

^K `wxRectangleShape ~wxRectangleShape`

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',

``wxrectangleshape')`)

^K `~wxRectangleShape`

`$#+K!wxRectangleShape::SetCornerRadius`

`void SetCornerRadius(double radius)K`

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.

`wxRectangleShape::SetCornerRadius`

`topic151`

`rowse00201`

`K wxRectangleShape SetCornerRadius`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxrectangleshape')")`

`K SetCornerRadius`

\$#+K! **wxShape::wxShape**

wxShape(wxShapeCanvas* *canvas* = *NULL*)^K

Constructs a new wxShape.

^wxShape::wxShape

^topic152

^browse00204

^K wxShape wxShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K wxShape

`$#+K!wxShape::~~wxShape`

`~wxShape()`^K

Destructor.

`wxShape::~~wxShape`
`topic153`
`browse00205`
`K wxShape ~wxShape`
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`
`K ~wxShape`

\$#+K! **wxShape::AddLine**

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

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::AddLine

^topic154

^browse00206

^K wxShape AddLine

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K AddLine

\$#+K! **wxShape::AddRegion**

void AddRegion(wxShapeRegion* *region*)^K

Adds a region to the shape.

^wxShape::AddRegion

^topic155

^browse00207

^K wxShape AddRegion

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K AddRegion

\$#+K! **wxShape::AddText**

void AddText(const wxString& *string*)^K

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

^wxShape::AddText

^topic156

^browse00208

^K wxShape AddText

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K AddText

\$#+K! **wxShape::AddToCanvas**

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

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

^wxShape::AddToCanvas

^topic157

^browse00209

^K wxShape AddToCanvas

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K AddToCanvas

`$#+KK!` **wxShape::AncestorSelected**

bool AncestorSelected() const

TRUE if the shape's ancestor is currently selected.

`wxShape::AncestorSelected`

`topic158`

`rowse00210`

`K wxShape AncestorSelected`

`K AncestorSelected`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

^{\$#+K!}**wxShape::ApplyAttachmentOrdering**

void ApplyAttachmentOrdering(wxList& *linesToSort*)^K

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

^wxShape::ApplyAttachmentOrdering

^wxshapeapplyattachmentordering

^browse00211

^K wxShape ApplyAttachmentOrdering

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K ApplyAttachmentOrdering

`$#+K!wxShape::AssignNewIds`

`void AssignNewIds()`^K

Assigns new ids to this image and its children.

`wxShape::AssignNewIds`

`topic159`

`browse00212`

`K wxShape AssignNewIds`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`K AssignNewIds`

\$#+K! **wxShape::Attach**

void Attach(wxShapeCanvas* *can*)^K

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

^wxShape::Attach

^wxshapeattach

^browse00213

^K wxShape Attach

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K Attach

`$#+KK!` **wxShape::AttachmentIsValid**

bool AttachmentIsValid(int *attachment*) const

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

`wxShape::AttachmentIsValid`

`wxshapeattachmentisvalid`

`rowse00214`

`K wxShape AttachmentIsValid`

`K AttachmentIsValid`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`$#+KKl 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 to determine the new line ordering.

`wxShape::AttachmentSortTest`

`wxshapeattachmentsorttest`

`rowse00215`

`K wxShape AttachmentSortTest`

`K AttachmentSortTest`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

\$#+K! **wxShape::CalcSimpleAttachment**

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

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::CalcSimpleAttachment

^wxshapecalcsimpleattachment

^browse00216

^K wxShape CalcSimpleAttachment

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K CalcSimpleAttachment

wxShape::CalculateSize

void CalculateSize()

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

wxShape::CalculateSize

topic160

rowse00217

wxShape CalculateSize

nableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

CalculateSize

`wxShape::ClearAttachments`

`void ClearAttachments()`^K

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

^w`xShape::ClearAttachments`

^t`opic161`

^b`rowse00218`

^K `wxShape ClearAttachments`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

^K `ClearAttachments`

`$#+K!wxShape::ClearRegions`

`void ClearRegions()`^K

Clears the wxShapeRegions from the shape.

`wxShape::ClearRegions`

`topic162`

`browse00219`

`K wxShape ClearRegions`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`K ClearRegions`

`wxShape::ClearText`

`void ClearText(int regionId = 0)`

Clears the text from the specified text region.

`wxShape::ClearText`

`topic163`

`rowse00220`

`wxShape ClearText`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`ClearText`

\$#+K! **wxShape::Constrain**

bool Constrain()^K

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

^wxShape::Constrain

^topic164

^browse00221

^K wxShape Constrain

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K Constrain

^{\$#+K!}**wxShape::Copy**

void Copy(wxShape& copy)^K

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.

^wxShape::Copy

^wxshapecopy

^browse00222

^K wxShape Copy

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K Copy

^{\$#+K!}**wxShape::CreateNewCopy**

wxShape* **CreateNewCopy**(**bool** *resetMapping* = *TRUE*, **bool** *recompute* = *TRUE*)^K

Creates and returns a new copy of this shape (calling wxShape::Copy). 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 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.

^wxShape::CreateNewCopy

^wxshapecreatenewcopy

^browse00223

^K wxShape CreateNewCopy

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K CreateNewCopy

\$#+K! **wxShape::DeleteControlPoints**

void DeleteControlPoints()^K

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

^wxShape::DeleteControlPoints

^topic165

^browse00224

^K wxShape DeleteControlPoints

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K DeleteControlPoints

\$#+K! **wxShape::Detach**

void Detach()^K

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

^wxShape::Detach

^topic166

^browse00225

^K wxShape Detach

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K Detach

\$#+K! **wxShape::Draggable**

bool Draggable()^K

TRUE if the shape may be dragged by the user.

wxShape::Draggable

topic167

browse00226

K wxShape Draggable

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

K Draggable

\$#+K! **wxShape::Draw**

void Draw(wxDC& dc)^K

Draws the whole shape and any lines attached to it.

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

^wxShape::Draw

^topic168

^browse00227

^K wxShape Draw

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K Draw

\$#+K! **wxShape::DrawContents**

void DrawContents(wxDC& dc)^K

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

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

^wxShape::DrawContents

^topic169

^browse00228

^K wxShape DrawContents

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K DrawContents

\$#+K! **wxShape::DrawLinks**

void DrawLinks(wxDC& *dc*, int *attachment* = -1)^K

Draws any lines linked to this shape.

^wxShape::DrawLinks

^topic170

^browse00229

^K wxShape DrawLinks

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K DrawLinks

`$#+K!wxShape::Erase`

`void Erase(wxDC& dc)K`

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

`wxShape::Erase`

`topic171`

`browse00230`

`K wxShape Erase`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`K Erase`

wxShape::EraseContents

void EraseContents(wxDC& dc)

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

wxShape::EraseContents

topic172

browse00231

wxShape EraseContents

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

EraseContents

^{\$#+K!}**wxShape::EraseLinks**

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

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

^wxShape::EraseLinks

^topic173

^browse00232

^K wxShape EraseLinks

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K EraseLinks

^{\$#+K!}**wxShape::FindRegion**

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

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

^wxShape::FindRegion

^topic174

^browse00233

^K wxShape FindRegion

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K FindRegion

^{\$#+K!}**wxShape::FindRegionNames**

void FindRegionNames(wxStringList& *list*)^K

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

^wxShape::FindRegionNames

^topic175

^browse00234

^K wxShape FindRegionNames

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K FindRegionNames

\$#+K!**wxShape::Flash**

void Flash()^K

Flashes the shape.

^wxShape::Flash
^topic176
^browse00235
^K wxShape Flash
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")
^K Flash

^{\$#+K!}**wxShape::FormatText**

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

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

^wxShape::FormatText

^topic177

^browse00236

^K wxShape FormatText

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K FormatText

`$#+KK!` **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](#).

`wxShape::GetAttachmentMode`

`topic178`

`browse00237`

`K wxShape GetAttachmentMode`

`K GetAttachmentMode`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

\$#+K! **wxShape::GetAttachmentPosition**

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

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.

^wwxShape::GetAttachmentPosition

^wwxshapegetattachmentposition

^browse00238

^K wxShape GetAttachmentPosition

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K GetAttachmentPosition

^{\$#+K!}**wxShape::GetBoundingBoxMax**

void GetBoundingBoxMax(double *width, double *height)^K

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

^wxShape::GetBoundingBoxMax

^topic179

^browse00239

^K wxShape GetBoundingBoxMax

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K GetBoundingBoxMax

^{\$#+K!}**wxShape::GetBoundingBoxMin**

void GetBoundingBoxMin(double *width, double *height)^K

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

^wxShape::GetBoundingBoxMin

^topic180

^browse00240

^K wxShape GetBoundingBoxMin

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K GetBoundingBoxMin

\$#+KK! **wxShape::GetBrush**

wxBrush* GetBrush() const

Returns the brush used for filling the shape.

wxShape::GetBrush

topic181

browse00241

K wxShape GetBrush

K GetBrush

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

\$#+KK! **wxShape::GetCanvas**

wxShapeCanvas* GetCanvas() const

Gets the internal canvas pointer.

wxShape::GetCanvas

topic182

browse00242

K wxShape GetCanvas

K GetCanvas

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

`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::GetCentreResize`

`topic183`

`rowse00243`

`wxShape GetCentreResize`

`GetCentreResize`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`$#+KKl wxShape::GetChildren`

`wxList& GetChildren() const`

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

`wxShape::GetChildren`

`topic184`

`rowse00244`

`K wxShape GetChildren`

`K GetChildren`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`$#+K! wxShape::GetClientData`

`wxObject* GetClientData()`^K

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

^wxShape::GetClientData

^topic185

^browse00245

^K wxShape GetClientData

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K GetClientData

`$#+KK!` **wxShape::GetDisableLabel**

bool GetDisableLabel() const

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

`wxShape::GetDisableLabel`

`topic186`

`browse00246`

`K wxShape GetDisableLabel`

`K GetDisableLabel`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

\$#+KK!**wxShape::GetEventHandler**

wxShapeEvtHandler* GetEventHandler() const

Returns the event handler for this shape.

wxShape::GetEventHandler

topic187

browse00247

K wxShape GetEventHandler

K GetEventHandler

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

`$#+KK!wxShape::GetFixedHeight`

`bool GetFixedHeight() const`

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

`wxShape::GetFixedHeight`

`topic188`

`rowse00248`

`K wxShape GetFixedHeight`

`K GetFixedHeight`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

^{\$#+K!}**wxShape::GetFixedSize**

void GetFixedSize(**bool** * *x*, **bool** * *y*)^K

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

^wxShape::GetFixedSize

^topic189

^browse00249

^K wxShape GetFixedSize

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K GetFixedSize

`$#+KK!wxShape::GetFixedWidth`

`bool GetFixedWidth() const`

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

`wxShape::GetFixedWidth`

`topic190`

`rowse00250`

`K wxShape GetFixedWidth`

`K GetFixedWidth`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

\$#+KK! **wxShape::GetFont**

wxFont* GetFont(int *regionId* = 0) const

Gets the font for the specified text region.

^wxShape::GetFont

^topic191

^browse00251

^K wxShape GetFont

^K GetFont

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

`$#+KK!` **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::GetFunctor`

`topic192`

`browse00252`

`K wxShape GetFunctor`

`K GetFunctor`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

\$#+KK!**wxShape::GetId**

long GetId() const

Returns the integer identifier for this shape.

wxShape::GetId

topic193

browse00253

K wxShape GetId

K GetId

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

\$#+K! **wxShape::GetLinePosition**

int GetLinePosition(wxLineShape* *line*)^K

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

^wxShape::GetLinePosition

^wxshapegetlineposition

^browse00254

^K wxShape GetLinePosition

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K GetLinePosition

\$#+KK! **wxShape::GetLines**

wxList& GetLines() const

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

wxShape::GetLines

topic194

browse00255

K wxShape GetLines

K GetLines

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

`$#+KK!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::GetMaintainAspectRatio`

`wxshapegetmaintainaspectratio`

`rowse00256`

`K wxShape GetMaintainAspectRatio`

`K GetMaintainAspectRatio`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

\$#+KK! **wxShape::GetNumberOfAttachments**

int GetNumberOfAttachments() const

Gets the number of attachment points for this shape.

^wxShape::GetNumberOfAttachments

^wxshapegetnumberofattachments

^browse00257

^K wxShape GetNumberOfAttachments

^K GetNumberOfAttachments

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

\$#+KK!**wxShape::GetNumberOfTextRegions**

int GetNumberOfTextRegions() const

Gets the number of text regions for this shape.

wxShape::GetNumberOfTextRegions

topic195

browse00258

K wxShape GetNumberOfTextRegions

K GetNumberOfTextRegions

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

\$#+KK! **wxShape::GetParent**

wxShape * GetParent() const

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

wxShape::GetParent

topic196

browse00259

K wxShape GetParent

K GetParent

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp`, `wxshape`)")

`$#+KKl` **wxShape::GetPen**

wxPen* GetPen() const

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

`wxShape::GetPen`

`topic197`

`browse00260`

`K wxShape GetPen`

`K GetPen`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`$#+K!` **wxShape::GetPerimeterPoint**

bool GetPerimeterPoint(double x1, double y1, double x2, double y2, double *x3, double *y3)^K

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

`w`xShape::GetPerimeterPoint

`t`opic198

`b`rowse00261

`K` wxShape GetPerimeterPoint

`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

`K` GetPerimeterPoint

^{\$#+K!}**wxShape::GetRegionId**

int GetRegionId(const wxString& name)^K

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

^wxShape::GetRegionId

^getregionid

^browse00262

^K wxShape GetRegionId

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K GetRegionId

\$#+K! **wxShape::GetRegionName**

wxString GetRegionName(int *regionId* = 0)^K

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](#).

^wxShape::GetRegionName

^getregionname

^browse00263

^K wxShape GetRegionName

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K GetRegionName

^{\$#+K!}**wxShape::GetRegions**

wxList& GetRegions()^K

Returns the list of wxShapeRegions.

^wxShape::GetRegions

^getregions

^browse00264

^K wxShape GetRegions

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K GetRegions

`wxShape::GetRotation`

`double GetRotation() const`

Returns the angle of rotation in radians.

`wxShape::GetRotation`

`topic199`

`rowse00265`

`wxShape GetRotation`

`GetRotation`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

wxShape::GetSensitivityFilter

void GetSensitivityFilter() const

Returns the sensitivity filter, a bitlist of values. See [wxShape::SetSensitivityFilter](#).

wxShape::GetSensitivityFilter

topic200

rowse00266

wxShape GetSensitivityFilter

GetSensitivityFilter

enableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

`wxShape::GetShadowMode`

`int SetShadowMode() const`

Returns the shadow mode. See [wxShape::SetShadowMode](#).

`wxShape::GetShadowMode`

`topic201`

`rowse00267`

`wxShape GetShadowMode`

`SetShadowMode`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`$#+KK!` **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::GetSpaceAttachments`

`topic202`

`rowse00268`

`K wxShape GetSpaceAttachments`

`K GetSpaceAttachments`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

\$#+KK! **wxShape::GetTextColour**

wxString GetTextColour(int *regionId* = 0) const

Gets the colour for the specified text region.

^wwxShape::GetTextColour

^topic203

^browse00269

^K wxShape GetTextColour

^K GetTextColour

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

\$#+KK!**wxShape::GetTopAncestor**

wxShape * GetTopAncestor() const

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

wxShape::GetTopAncestor

topic204

browse00270

K wxShape GetTopAncestor

K GetTopAncestor

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

`$#+KKl wxShape::GetX`

`double GetX() const`

Gets the x position of the centre of the shape.

`wxShape::GetX`

`topic205`

`browse00271`

`K wxShape GetX`

`K GetX`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`$#+KKl wxShape::GetY`

`double GetY() const`

Gets the y position of the centre of the shape.

`wxShape::GetY`

`topic206`

`rowse00272`

`K wxShape GetY`

`K GetY`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`$#+K!` **wxShape::HitTest**

bool HitTest(double x, double y, int* attachment, double* distance)^K

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::HitTest`

`topic207`

`browse00273`

`K wxShape HitTest`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`K HitTest`

^{\$#+K!}**wxShape::Insert**

void InsertInCanvas(wxShapeCanvas* *canvas*)^K

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

^wxShape::Insert

^topic208

^browse00274

^K wxShape Insert

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K InsertInCanvas

`$#+KKl` **wxShape::IsHighlighted**

bool IsHighlighted() const

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

`wxShape::IsHighlighted`

`topic209`

`browse00275`

`K wxShape IsHighlighted`

`K IsHighlighted`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`$#+KK!` **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::IsShown`

`topic210`

`rowse00276`

`K wxShape IsShown`

`K IsShown`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

^{\$#+K!}**wxShape::MakeControlPoints**

void MakeControlPoints()^K

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

^wxShape::MakeControlPoints

^topic211

^browse00277

^K wxShape MakeControlPoints

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K MakeControlPoints

^{\$#+K!}**wxShape::MakeMandatoryControlPoints**

void MakeMandatoryControlPoints()^K

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::MakeMandatoryControlPoints

^topic212

^browse00278

^K wxShape MakeMandatoryControlPoints

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K MakeMandatoryControlPoints

\$#+K! **wxShape::Move**

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

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

^wxShape::Move

^wxshapemove

^browse00279

^K wxShape Move

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K Move

`$#+K! wxShape::MoveLineToNewAttachment`

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

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.

Cals `wxShape::AttachmentSortTest` and then `wxShapeEvtHandler::OnChangeAttachment`.

`wxShape::MoveLineToNewAttachment`

`wxshapemovelinetonewattachment`

`rowse00280`

`K wxShape MoveLineToNewAttachment`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`K MoveLineToNewAttachment`

^{\$#+K!}**wxShape::MoveLinks**

void MoveLinks(wxDC& dc)^K

Redraw all the lines attached to the shape.

^wxShape::MoveLinks

^topic213

^browse00281

^K wxShape MoveLinks

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K MoveLinks

^{\$#+K!}**wxShape::NameRegions**

void NameRegions(const wxString& *parentName* = "")^K

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

^wxShape::NameRegions

^topic214

^browse00282

^K wxShape NameRegions

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K NameRegions

\$#+K! **wxShape::Rotate**

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

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::Rotate

^topic215

^browse00283

^K wxShape Rotate

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K Rotate

^{\$#+K!}**wxShape::ReadConstraints**

void ReadConstraints(**wxExpr** **clause*, **wxExprDatabase** **database*)^K

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

^wxShape::ReadConstraints

^topic216

^browse00284

^K wxShape ReadConstraints

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K ReadConstraints

\$#+K! **wxShape::ReadAttributes**

void ReadAttributes(wxExpr* *clause*)^K

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

^wxShape::ReadAttributes

^topic217

^browse00285

^K wxShape ReadAttributes

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K ReadAttributes

^{\$#+K!}**wxShape::ReadRegions**

void ReadRegions(wxExpr **clause*)^K

Reads in the regions.

^wxShape::ReadRegions

^topic218

^browse00286

^K wxShape ReadRegions

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K ReadRegions

^{\$#+K!}**wxShape::Recentre**

void Recentre()^K

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

^wxShape::Recentre

^topic219

^browse00287

^K wxShape Recentre

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K Recentre

^{\$#+K!}**wxShape::RemoveFromCanvas**

void RemoveFromCanvas(wxShapeCanvas* *canvas*)^K

Removes the shape from the canvas.

^wxShape::RemoveFromCanvas

^topic220

^browse00288

^K wxShape RemoveFromCanvas

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K RemoveFromCanvas

\$#+K! **wxShape::ResetControlPoints**

void ResetControlPoints()^K

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

^wxShape::ResetControlPoints

^topic221

^browse00289

^K wxShape ResetControlPoints

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K ResetControlPoints

\$#+K! **wxShape::ResetMandatoryControlPoints**

void ResetMandatoryControlPoints()^K

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::ResetMandatoryControlPoints

^topic222

^browse00290

^K wxShape ResetMandatoryControlPoints

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K ResetMandatoryControlPoints

\$#+K! **wxShape::Recompute**

bool Recompute()^K

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

^wxShape::Recompute
^wxshaperecompute
^browse00291
^K wxShape Recompute
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")
^K Recompute

\$#+K! **wxShape::RemoveLine**

void RemoveLine(wxLineShape* *line*)^K

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

^wxShape::RemoveLine

^topic223

^browse00292

^K wxShape RemoveLine

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K RemoveLine

\$#+K! **wxShape::Select**

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

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

^wxShape::Select

^wxshapeselect

^browse00293

^K wxShape Select

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K Select

`$#+KK!` **wxShape::Selected**

bool Selected() const

TRUE if the shape is currently selected.

`wxShape::Selected`

`wxshapeselected`

`rowse00294`

`K wxShape Selected`

`K Selected`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

\$#+K! **wxShape::SetAttachmentMode**

void SetAttachmentMode(**bool** *flag*)^K

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::SetAttachmentMode

^wxshap esetattachmentmode

^browse00295

^K wxShape SetAttachmentMode

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetAttachmentMode

\$#+K! **wxShape::SetBrush**

void SetBrush(wxBrush *brush)^K

Sets the brush for filling the shape's shape.

^wxShape::SetBrush

^topic224

^browse00296

^K wxShape SetBrush

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetBrush

^{\$#+K!}**wxShape::SetCanvas**

void SetCanvas(wxShapeCanvas* *theCanvas*)^K

Identical to wxShape::Attach.

^wxShape::SetCanvas

^wxshapесetcanvas

^browse00297

^K wxShape SetCanvas

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetCanvas

\$#+K! **wxShape::SetCentreResize**

void SetCentreResize(**bool** *cr*)^K

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::SetCentreResize

^topic225

^browse00298

^K wxShape SetCentreResize

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetCentreResize

\$#+K! **wxShape::SetClientData**

void SetClientData(wxObject *clientData)^K

Sets the client data.

^wxShape::SetClientData

^topic226

^browse00299

^K wxShape SetClientData

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetClientData

`$#+K! wxShape::SetDefaultRegionSize`

`void SetDefaultRegionSize()`^K

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

`wxShape::SetDefaultRegionSize`

`setdefaultregionsize`

`rowse00300`

`K wxShape SetDefaultRegionSize`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`K SetDefaultRegionSize`

`$#+K!wxShape::SetDisableLabel`

`void SetDisableLabel(bool flag)K`

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

`wxShape::SetDisableLabel`

`topic227`

`browse00301`

`K wxShape SetDisableLabel`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`K SetDisableLabel`

`$#+K!wxShape::SetDraggable`

`void SetDraggable(bool drag, bool recursive = FALSE)K`

Sets the shape to be draggable or not draggable.

`wxShape::SetDraggable`

`topic228`

`browse00302`

`K wxShape SetDraggable`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`K SetDraggable`

\$#+K! **wxShape::SetDrawHandles**

void SetDrawHandles(**bool** *drawH*)^K

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::SetDrawHandles

^topic229

^browse00303

^K wxShape SetDrawHandles

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetDrawHandles

^{\$#+K!}**wxShape::SetEventHandler**

void GetEventHandler(wxShapeEvtHandler *handler)^K

Sets the event handler for this shape.

^wxShape::SetEventHandler

^topic230

^browse00304

^K wxShape SetEventHandler

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K GetEventHandler

`$#+K!` **wxShape::SetFixedSize**

void SetFixedSize(bool *x*, bool *y*)^K

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

^wxShape::SetFixedSize

^topic231

^browse00305

^K wxShape SetFixedSize

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetFixedSize

\$#+K! **wxShape::SetFont**

void SetFont(wxFont *font, int regionId = 0)^K

Sets the font for the specified text region.

^wxShape::SetFont

^topic232

^browse00306

^K wxShape SetFont

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetFont

\$#+K! **wxShape::SetFormatMode**

void SetFormatMode(**int** *mode*, **int** *regionId* = 0)^K

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::SetFormatMode

setformatmode

browse00307

K wxShape SetFormatMode

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

K SetFormatMode

^{\$#+K!}**wxShape::SetHighlight**

void SetHighlight(**bool** *hi*, **bool** *recurse* = *FALSE*)^K

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

^wxShape::SetHighlight

^topic233

^browse00308

^K wxShape SetHighlight

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetHighlight

\$#+K!**wxShape::SetId**

void SetId(long *id*)^K

Set the integer identifier for this shape.

^wxShape::SetId
^topic234
^browse00309
^K wxShape SetId
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")
^K SetId

`$#+K!` **wxShape::SetMaintainAspectRatio**

void SetMaintainAspectRatio(**bool** *flag*)^K

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).

`w`xShape::SetMaintainAspectRatio

`w`xshapetmaintainaspectratio

`b`rowse00310

`K` wxShape SetMaintainAspectRatio

`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

`K` SetMaintainAspectRatio

\$#+K! **wxShape::SetPen**

void SetPen(wxPen *pen)^K

Sets the pen for drawing the shape's outline.

^wxShape::SetPen

^topic235

^browse00311

^K wxShape SetPen

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetPen

\$#+K! **wxShape::SetRegionName**

void SetRegionName(const wxString& *name*, int *regionId* = 0)^K

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::SetRegionName

^wxshapeseendregionname

^browse00312

^K wxShape SetRegionName

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetRegionName

^{\$#+K!}**wxShape::SetSensitivityFilter**

void SetSensitivityFilter(int *sens*=*OP_ALL*, bool *recursive* = *FALSE*)^K

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

sens is a bitlist of the following:

{bmc bullet.bmp} OP_CLICK_LEFT

{bmc bullet.bmp} OP_CLICK_RIGHT

{bmc bullet.bmp} OP_DRAG_LEFT

{bmc bullet.bmp} OP_DRAG_RIGHT

{bmc bullet.bmp} OP_ALL (equivalent to a combination of all the above).

^wxShape::SetSensitivityFilter

^wxshapesetsensitivityfilter

^browse00313

^K wxShape SetSensitivityFilter

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetSensitivityFilter

\$#+K! **wxShape::SetShadowMode**

void SetShadowMode(**int** *mode*, **bool** *redraw* = *FALSE*)^K

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.

^wwxShape::SetShadowMode

^wxshapersetshadowmode

^browse00314

^K wxShape SetShadowMode

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetShadowMode

\$#+K! **wxShape::SetSize**

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

Sets the shape's size.

^wxShape::SetSize

^topic236

^browse00315

^K wxShape SetSize

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetSize

`$#+K!` **wxShape::SetSpaceAttachments**

void SetSpaceAttachments(bool *sp*)^K

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::SetSpaceAttachments

^topic237

^browse00316

^K wxShape SetSpaceAttachments

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetSpaceAttachments

`wxShape::SetTextColour`

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

Sets the colour for the specified text region.

`wxShape::SetTextColour`

`topic238`

`rowse00317`

`wxShape SetTextColour`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`SetTextColour`

\$#+K! **wxShape::SetX**

void SetX(double x)^K

Sets the x position of the shape.

^wxShape::SetX

^topic239

^browse00318

^K wxShape SetX

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetX

\$#+K! **wxShape::SetX**

void SetY(double y)^K

Sets the y position of the shape.

^wxShape::SetX

^topic240

^browse00319

^K wxShape SetX

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SetY

\$#+K! **wxShape::SpaceAttachments**

void SpaceAttachments(**bool** *sp*)^K

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::SpaceAttachments

^topic241

^browse00320

^K wxShape SpaceAttachments

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K SpaceAttachments

\$#+K! **wxShape::Show**

void Show(**bool** *show*)^K

Sets a flag indicating whether the shape should be drawn.

^wxShape::Show

^topic242

^browse00321

^K wxShape Show

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K Show

`$#+K!wxShape::Unlink`

`void Unlink()`^K

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::Unlink`

`topic243`

`browse00322`

`K wxShape Unlink`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")`

`K Unlink`

\$#+K! **wxShape::WriteAttributes**

void WriteAttributes(wxExpr **clause*)^K

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

^wxShape::WriteAttributes

^topic244

^browse00323

^K wxShape WriteAttributes

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K WriteAttributes

^{\$#+K!}**wxShape::WriteRegions**

void WriteRegions(wxExpr **clause*)^K

Writes the regions.

^wxShape::WriteRegions

^topic245

^browse00324

^K wxShape WriteRegions

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshape')")

^K WriteRegions

^{\$#+K!}**wxShapeCanvas::wxShapeCanvas**

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

Constructor.

^wxShapeCanvas::wxShapeCanvas

^topic246

^browse00326

^K wxShapeCanvas wxShapeCanvas

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K wxShapeCanvas

`$#+K! wxShapeCanvas::~~wxShapeCanvas`

`~wxShapeCanvas()`^K

Destructor.

^wxShapeCanvas::~~wxShapeCanvas

^topic247

^browse00327

^K wxShapeCanvas ~wxShapeCanvas

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K ~wxShapeCanvas

`$#+K!wxShapeCanvas::AddShape`

`void AddShape(wxShape *shape, wxShape *addAfter = NULL)K`

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

`wxShapeCanvas::AddShape`

`topic248`

`browse00328`

`K wxShapeCanvas AddShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")`

`K AddShape`

\$#+K! **wxShapeCanvas::FindShape**

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

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::FindShape

topic249

browse00329

K wxShapeCanvas FindShape

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

K FindShape

^{\$#+K!}**wxShapeCanvas::FindFirstSensitiveShape**

wxShape * FindFirstSensitiveShape(double x1, double y, int *attachment, int op)^K

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

^wxShapeCanvas::FindFirstSensitiveShape

^topic250

^browse00330

^K wxShapeCanvas FindFirstSensitiveShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K FindFirstSensitiveShape

`$#+KK!wxShapeCanvas::GetDiagram`

`wxDiagram* GetDiagram() const`

Returns the canvas associated with this diagram.

`wxShapeCanvas::GetDiagram`

`topic251`

`rowse00331`

`K wxShapeCanvas GetDiagram`

`K GetDiagram`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")`

`$#+KK!` **wxShapeCanvas::GetGridSpacing**

double GetGridSpacing() const

Returns the grid spacing.

`wxShapeCanvas::GetGridSpacing`

`topic252`

`rowse00332`

`K wxShapeCanvas GetGridSpacing`

`K GetGridSpacing`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")`

`$#+KK!`**wxShapeCanvas::GetMouseTolerance**

int GetMouseTolerance() const

Returns the tolerance within which a mouse move is ignored.

`wxShapeCanvas::GetMouseTolerance`

`topic253`

`rowse00333`

`K wxShapeCanvas GetMouseTolerance`

`K GetMouseTolerance`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")`

`$#+KK!wxShapeCanvas::GetShapeList`

`wxList* GetShapeList() const`

Returns a pointer to the internal shape list.

`wxShapeCanvas::GetShapeList`

`topic254`

`rowse00334`

`K wxShapeCanvas GetShapeList`

`K GetShapeList`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")`

`$#+KK!` **wxShapeCanvas::GetQuickEditMode**

bool GetQuickEditMode() const

Returns quick edit mode for the associated diagram.

`wxShapeCanvas::GetQuickEditMode`

`topic255`

`rowse00335`

`K wxShapeCanvas GetQuickEditMode`

`K GetQuickEditMode`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")`

\$#+K! **wxShapeCanvas::InsertShape**

void InsertShape(wxShape* *shape*)^K

Inserts a shape at the front of the shape list.

^wxShapeCanvas::InsertShape

^topic256

^browse00336

^K wxShapeCanvas InsertShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K InsertShape

^{\$#+K!}**wxShapeCanvas::OnBeginDragLeft**

void OnBeginDragLeft(double x, double y, int keys = 0)^K

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:

{bmc bullet.bmp} KEY_SHIFT

{bmc bullet.bmp} KEY_CTRL

See also [wxShapeCanvas::OnDragLeft](#), [wxShapeCanvas::OnEndDragLeft](#).

^wxShapeCanvas::OnBeginDragLeft

^wxshapecanvasonbegindragleft

^browse00337

^K wxShapeCanvas OnBeginDragLeft

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K OnBeginDragLeft

\$#+K! **wxShapeCanvas::OnBeginDragRight**

void OnBeginDragRight(double x, double y, int keys = 0)^K

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:

{bmc bullet.bmp} KEY_SHIFT

{bmc bullet.bmp} KEY_CTRL

See also [wxShapeCanvas::OnDragRight](#), [wxShapeCanvas::OnEndDragRight](#).

^wxShapeCanvas::OnBeginDragRight

^wxshapecanvasonbegindragright

^browse00338

^K wxShapeCanvas OnBeginDragRight

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K OnBeginDragRight

\$#+K! **wxShapeCanvas::OnEndDragLeft**

void OnEndDragLeft(double x, double y, int keys = 0)^K

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:

{bmc bullet.bmp} KEY_SHIFT

{bmc bullet.bmp} KEY_CTRL

See also [wxShapeCanvas::OnDragLeft](#), [wxShapeCanvas::OnBeginDragLeft](#).

^wxShapeCanvas::OnEndDragLeft

^wxshapecanvasonenddragleft

^browse00339

^K wxShapeCanvas OnEndDragLeft

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K OnEndDragLeft

\$#+K! **wxShapeCanvas::OnEndDragRight**

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

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:

{bmc bullet.bmp} KEY_SHIFT

{bmc bullet.bmp} KEY_CTRL

See also [wxShapeCanvas::OnDragRight](#), [wxShapeCanvas::OnBeginDragRight](#).

^wxShapeCanvas::OnEndDragRight

^wxshapecanvasonenddragright

^browse00340

^K wxShapeCanvas OnEndDragRight

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K OnEndDragRight

\$#+K! **wxShapeCanvas::OnDragLeft**

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

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:

{bmc bullet.bmp} KEY_SHIFT

{bmc bullet.bmp} KEY_CTRL

See also [wxShapeCanvas::OnBeginDragLeft](#), [wxShapeCanvas::OnEndDragLeft](#).

^wxShapeCanvas::OnDragLeft

^wxshapecanvasondragleft

^browse00341

^K wxShapeCanvas OnDragLeft

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K OnDragLeft

\$#+K! **wxShapeCanvas::OnDragRight**

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

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:

{bmc bullet.bmp} KEY_SHIFT

{bmc bullet.bmp} KEY_CTRL

See also [wxShapeCanvas::OnBeginDragRight](#), [wxShapeCanvas::OnEndDragRight](#).

^wxShapeCanvas::OnDragRight

^wxshapecanvasondragright

^browse00342

^K wxShapeCanvas OnDragRight

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K OnDragRight

\$#+K! **wxShapeCanvas::OnLeftClick**

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

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:

{bmc bullet.bmp} KEY_SHIFT

{bmc bullet.bmp} KEY_CTRL

^wxShapeCanvas::OnLeftClick

^wxshapecanvasonleftclick

^browse00343

^K wxShapeCanvas OnLeftClick

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K OnLeftClick

`wxShapeCanvas::OnRightClick`

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

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:

{bmc bullet.bmp} KEY_SHIFT

{bmc bullet.bmp} KEY_CTRL

^wxShapeCanvas::OnRightClick

^wxshapecanvasonrightclick

^browse00344

^K wxShapeCanvas OnRightClick

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K OnRightClick

`$#+K! wxShapeCanvas::Redraw`

`void Redraw()`^K

Calls wxDiagram::Redraw.

`wxShapeCanvas::Redraw`

`topic257`

`browse00345`

`K wxShapeCanvas Redraw`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")`

`K Redraw`

^{\$#+K!}**wxShapeCanvas::RemoveShape**

void RemoveShape(wxShape **shape*)^K

Calls wxDiagram::RemoveShape.

^wxShapeCanvas::RemoveShape

^topic258

^browse00346

^K wxShapeCanvas RemoveShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K RemoveShape

\$#+K! **wxShapeCanvas::SetDiagram**

void SetDiagram(wxDiagram **diagram*)^K

Sets the diagram associated with this diagram.

^wxShapeCanvas::SetDiagram

^topic259

^browse00347

^K wxShapeCanvas SetDiagram

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K SetDiagram

\$#+K! **wxShapeCanvas::Snap**

void Snap(double *x, double *y)^K

Calls wxDiagram::Snap.

^wxShapeCanvas::Snap

^topic260

^browse00348

^K wxShapeCanvas Snap

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxshapecanvas')")

^K Snap

^{\$#+K!}**wxShapeEvtHandler::m_handlerShape**

wxShape* m_handlerShape^K

Pointer to the shape associated with this handler.

^wxShapeEvtHandler::m_handlerShape

^topic261

^browse00350

^K wxShapeEvtHandler m_handlerShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

^K m_handlerShape

^{\$#+K!}**wxShapeEvtHandler::m_previousHandler**

wxShapeEvtHandler* m_previousHandler^K

Pointer to the previous handler.

^wxShapeEvtHandler::m_previousHandler

^topic262

^browse00351

^K wxShapeEvtHandler m_previousHandler

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

^K m_previousHandler

`wxShapeEvtHandler::wxShapeEvtHandler`

`void wxShapeEvtHandler(wxShapeEvtHandler *previous = NULL, wxShape *shape = NULL)`^K

Constructs a new event handler.

`wxShapeEvtHandler::wxShapeEvtHandler`

`topic263`

`browse00352`

`wxShapeEvtHandler wxShapeEvtHandler`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler'))`

`wxShapeEvtHandler`

`$#+K!wxShapeEvtHandler::~~wxShapeEvtHandler`

`void ~wxShapeEvtHandler()`^K

Destructor.

`wxShapeEvtHandler::~~wxShapeEvtHandler`

`topic264`

`browse00353`

`K wxShapeEvtHandler ~wxShapeEvtHandler`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")`

`K ~wxShapeEvtHandler`

wxShapeEvtHandler::CopyData

void CopyData(wxShapeEvtHandler& *handler*)^K

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

^wxShapeEvtHandler::CopyData
^wxshapeevthandlerycopydata
^browse00354
^K wxShapeEvtHandler CopyData
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler'))
^K CopyData

^{\$#+K!}**wxShapeEvtHandler::CreateNewCopy**

wxShapeEvtHandler* CreateNewCopy()^K

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

^wxShapeEvtHandler::CreateNewCopy

^wxshapeevthandlercreatenewcopy

^browse00355

^K wxShapeEvtHandler CreateNewCopy

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

^K CreateNewCopy

`$#+KK!` **wxShapeEvtHandler::GetPreviousHandler**

wxShapeEvtHandler* GetPreviousHandler() const

Returns the previous handler.

`wxShapeEvtHandler::GetPreviousHandler`

`wxshapeevthandlergetprevioushandler`

`rowse00356`

`K wxShapeEvtHandler GetPreviousHandler`

`K GetPreviousHandler`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp',
`wxshapeevthandler')")`

`wxShapeEvtHandler::GetShape`

`wxShape* GetShape() const`

Returns the shape associated with this handler.

`wxShapeEvtHandler::GetShape`

`wxshapeevthandlergetshape`

`rowse00357`

`wxShapeEvtHandler GetShape`

`GetShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp',
`wxshapeevthandler')")`

wxShapeEvtHandler::OnBeginDragLeft

void OnBeginDragLeft(double x, double y, int keys=0, int attachment = 0)^K

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

^wxShapeEvtHandler::OnBeginDragLeft

^topic265

^browse00358

^K wxShapeEvtHandler OnBeginDragLeft

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

^K OnBeginDragLeft

`wxShapeEvtHandler::OnBeginDragRight`

`void OnBeginDragRight(double x, double y, int keys=0, int attachment = 0)`^K

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

^w`wxShapeEvtHandler::OnBeginDragRight`

^t`opic266`

^b`rowse00359`

^K `wxShapeEvtHandler OnBeginDragRight`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")`

^K `OnBeginDragRight`

\$#+K! **wxShapeEvtHandler::OnBeginSize**

void OnBeginSize(double *width*, double *height*)^K

Called when a shape starts to be resized.

wxShapeEvtHandler::OnBeginSize

topic267

browse00360

K wxShapeEvtHandler OnBeginSize

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

K OnBeginSize

^{##+K!}**wxShapeEvtHandler::OnChangeAttachment**

void OnChangeAttachment(int *attachment*, wxLineShape* *line*, wxList& *ordering*)^K

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

^wxShapeEvtHandler::OnChangeAttachment

^wxshapeevthandleronchangeattachment

^browse00361

^K wxShapeEvtHandler OnChangeAttachment

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

^K OnChangeAttachment

\$#+K! **wxShapeEvtHandler::OnDragLeft**

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

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::OnDragLeft

topic268

browse00362

K wxShapeEvtHandler OnDragLeft

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

K OnDragLeft

wxShapeEvtHandler::OnDragRight

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

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

^wwxShapeEvtHandler::OnDragRight

^topic269

^browse00363

^K wxShapeEvtHandler OnDragRight

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

^K OnDragRight

`$#+K! wxShapeEvtHandler::OnDraw`

`void OnDraw(wxDC& dc)K`

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

`wxShapeEvtHandler::OnDraw`

`topic270`

`browse00364`

`K wxShapeEvtHandler OnDraw`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',`

``wxshapeevthandler'))`

`K OnDraw`

wxShapeEvtHandler::OnDrawContents

void OnDrawContents(wxDC& dc)

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

wxShapeEvtHandler::OnDrawContents

topic271

browse00365

wxShapeEvtHandler OnDrawContents

enableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp",

`wxshapeevthandler'))

OnDrawContents

wxShapeEvtHandler::OnDrawControlPoints

void OnDrawControlPoints(wxDC& dc)^K

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

^wxShapeEvtHandler::OnDrawControlPoints

^topic272

^browse00366

^K wxShapeEvtHandler OnDrawControlPoints

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

^K OnDrawControlPoints

wxShapeEvtHandler::OnDrawOutline

void OnDrawOutline(wxDC& dc)^K

Called when the outline of the shape should be drawn.

^wxShapeEvtHandler::OnDrawOutline

^topic273

^browse00367

^K wxShapeEvtHandler OnDrawOutline

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

^K OnDrawOutline

\$#+K! **wxShapeEvtHandler::OnEndDragLeft**

void OnEndDragLeft(double x, double y, int keys=0, int attachment = 0)^K

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

wxShapeEvtHandler::OnEndDragLeft

topic274

browse00368

K wxShapeEvtHandler OnEndDragLeft

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

K OnEndDragLeft

^{\$#+K!}**wxShapeEvtHandler::OnEndDragRight**

void OnEndDragRight(double *x*, double *y*, int *keys*=0, int *attachment* = 0)^K

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

^wxShapeEvtHandler::OnEndDragRight

^topic275

^browse00369

^K wxShapeEvtHandler OnEndDragRight

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',

[`]wxshapeevthandler')")

^K OnEndDragRight

`wxShapeEvtHandler::OnEndSize`

`void OnEndSize(double width, double height)`

Called after a shape is resized.

`wxShapeEvtHandler::OnEndSize`

`topic276`

`rowse00370`

`wxShapeEvtHandler OnEndSize`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp",
`wxshapeevthandler')")`

`OnEndSize`

wxShapeEvtHandler::OnErase

void OnErase(wxDC& dc)

Called when the whole shape should be erased.

wxShapeEvtHandler::OnErase

topic277

rowse00371

wxShapeEvtHandler OnErase

enableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp',
`wxshapeevthandler')")

OnErase

wxShapeEvtHandler::OnEraseContents

void OnEraseContents(wxDC& dc)^K

Called when the contents should be erased.

^wxShapeEvtHandler::OnEraseContents

^topic278

^browse00372

^K wxShapeEvtHandler OnEraseContents

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

^K OnEraseContents

`wxShapeEvtHandler::OnEraseControlPoints`

`void OnEraseControlPoints(wxDC& dc)`^K

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

^w`wxShapeEvtHandler::OnEraseControlPoints`

^t`opic279`

^b`rowse00373`

^K `wxShapeEvtHandler OnEraseControlPoints`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")`

^K `OnEraseControlPoints`

wxShapeEvtHandler::OnHighlight

void OnHighlight(wxDC& dc)

Called when the shape should be highlighted.

wxShapeEvtHandler::OnHighlight

topic280

rowse00374

wxShapeEvtHandler OnHighlight

enableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp',
`wxshapeevthandler')")

OnHighlight

wxShapeEvtHandler::OnLeftClick

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

Called when the shape receives a left mouse click event.

^wxShapeEvtHandler::OnLeftClick

^topic281

^browse00375

^K wxShapeEvtHandler OnLeftClick

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

^K OnLeftClick

^{\$#+K!}**wxShapeEvtHandler::OnMoveLink**

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

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

^wxShapeEvtHandler::OnMoveLink

^topic282

^browse00376

^K wxShapeEvtHandler OnMoveLink

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler'))

^K OnMoveLink

wxShapeEvtHandler::OnMoveLinks

void OnMoveLinks(wxDC& dc)

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

wxShapeEvtHandler::OnMoveLinks

topic283

rowse00377

wxShapeEvtHandler OnMoveLinks

**enableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp',
`wxshapeevthandler')")**

OnMoveLinks

`wxShapeEvtHandler::OnMovePost`

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

Called just after the shape receives a move request.

^w`wxShapeEvtHandler::OnMovePost`

^t`opic284`

^b`rowse00378`

^K `wxShapeEvtHandler OnMovePost`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler'))"`

^K `OnMovePost`

\$#+K! **wxShapeEvtHandler::OnMovePre**

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

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

wxShapeEvtHandler::OnMovePre

topic285

browse00379

K wxShapeEvtHandler OnMovePre

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")

K OnMovePre

`wxShapeEvtHandler::OnRightClick`

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

Called when the shape receives a mouse mouse click event.

^w`wxShapeEvtHandler::OnRightClick`

^t`opic286`

^b`rowse00380`

^K `wxShapeEvtHandler OnRightClick`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")`

^K `OnRightClick`

wxShapeEvtHandler::OnSize

void OnSize(double x, double y)^K

Called when the shape receives a resize request.

^wxShapeEvtHandler::OnSize

^topic287

^browse00381

^K wxShapeEvtHandler OnSize

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',

[`]wxshapeevthandler'))

^K OnSize

`wxShapeEvtHandler::OnSizingBeginDragLeft`

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

Called when a sizing drag is beginning.

`wxShapeEvtHandler::OnSizingBeginDragLeft`
^topic288
^browse00382
^K wxShapeEvtHandler OnSizingBeginDragLeft
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")
^K OnSizingBeginDragLeft

\$#+K! **wxShapeEvtHandler::OnSizingDragLeft**

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

Called when a sizing drag is occurring.

^wwxShapeEvtHandler::OnSizingDragLeft

^topic289

^browse00383

^K wxShapeEvtHandler OnSizingDragLeft

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',

[`]wxshapeevthandler')")

^K OnSizingDragLeft

\$#+K! **wxShapeEvtHandler::OnSizingEndDragLeft**

void OnSizingEndDragLeft(wxCtrlPoint* pt, double x, double y, int keys=0, int attachment = 0)^K

Called when a sizing drag is ending.

wxShapeEvtHandler::OnSizingEndDragLeft
topic290
browse00384
K wxShapeEvtHandler OnSizingEndDragLeft
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp',
`wxshapeevthandler')")
K OnSizingEndDragLeft

\$#+K! **wxShapeEvtHandler::SetPreviousHandler**

void SetPreviousHandler(wxShapeEvtHandler* *handler*)^K

Sets the previous handler.

^wxShapeEvtHandler::SetPreviousHandler

^wxshapeevthandlersetprevioushandler

^browse00385

^K wxShapeEvtHandler SetPreviousHandler

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ogl.hlp",

`wxshapeevthandler'))

^K SetPreviousHandler

`wxShapeEvtHandler::SetShape`

`void SetShape(wxShape* shape)`

Sets the shape for this handler.

`wxShapeEvtHandler::SetShape`

`wxshapeevthandlersetshape`

`rowse00386`

`wxShapeEvtHandler SetShape`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp',
`wxshapeevthandler')")`

`SetShape`

`$#+K!wxTextShape::wxTextShape`

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

Constructor.

`wxTextShape::wxTextShape`

`topic291`

`rowse00388`

`K wxTextShape wxTextShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxttextshape')")`

`K wxTextShape`

`$#+K!wxTextShape::~~wxTextShape`

`void ~wxTextShape()`^K

Destructor.

`wxTextShape::~~wxTextShape`

`topic292`

`browse00389`

`K wxTextShape ~wxTextShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxttextshape')")`

`K ~wxTextShape`

`$#+K!::wxOGLInitialize`

`void wxOGLInitialize()`^K Initializes OGL.

`::wxOGLInitialize`

`topic293`

`browse00391`

`K wxOGLInitialize`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `functions')")`

`K wxOGLInitialize`

`$#+K!::wxOGLCleanUp`

`void wxOGLCleanUp()`^K Cleans up OGL.

`::wxOGLCleanUp`

`topic294`

`browse00392`

`K wxOGLCleanUp`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `functions')")`

`K wxOGLCleanUp`

