```
#pragma once
#ifndef PIP_DEMO_UTILITIES_INCLUDED
 #include "DemoUtilities.h"
#endif

//
    ========================================================================
    ======
static void showBubbleMessage (Component& targetComponent, const String&
    textToShow,
                               std::unique_ptr<BubbleMessageComponent>& bmc,
                               bool isRunningComponentTransformDemo);

//
    ========================================================================
    ======
/** To demonstrate how sliders can have custom snapping applied to their
    values,
    this simple class snaps the value to 50 if it comes near.
*/
struct SnappingSlider  : public Slider
{
    double snapValue (double attemptedValue, DragMode dragMode) override
    {
        if (dragMode == notDragging)
            return attemptedValue;  // if they're entering the value in the
                text-box, don't mess with it.

        if (attemptedValue > 40 && attemptedValue < 60)
            return 50.0;

        return attemptedValue;
    }
};

/** A TextButton that pops up a colour chooser to change its colours. */
class ColourChangeButton  : public TextButton,
                            public ChangeListener
{
public:
    ColourChangeButton()
        : TextButton ("Click to change colour...")
    {
        setSize (10, 24);
        changeWidthToFitText();
    }

    void clicked() override
    {
        auto* colourSelector = new ColourSelector();
        colourSelector->setName ("background");
        colourSelector->setCurrentColour (findColour (TextButton::
            buttonColourId));
        colourSelector->addChangeListener (this);
        colourSelector->setColour (ColourSelector::backgroundColourId,
            Colours::transparentBlack);
        colourSelector->setSize (300, 400);

        CallOutBox::launchAsynchronously (colourSelector, getScreenBounds(),
            nullptr);
    }

    void changeListenerCallback (ChangeBroadcaster* source) override
    {
        if (auto* cs = dynamic_cast<ColourSelector*> (source))
            setColour (TextButton::buttonColourId, cs->getCurrentColour());
    }
};

//
    ==========================================================================
    ======
struct SlidersPage  : public Component
{
    SlidersPage()
    {
        Rectangle<int> layoutArea { 20, 20, 580, 430 };
        auto sliderArea = layoutArea.removeFromTop (320);

        auto* s = createSlider (false);
        s->setSliderStyle (Slider::LinearVertical);
        s->setTextBoxStyle (Slider::TextBoxBelow, false, 100, 20);
        s->setBounds (sliderArea.removeFromLeft (70));
        s->setDoubleClickReturnValue (true, 50.0); // double-clicking this
            slider will set it to 50.0
        s->setTextValueSuffix (" units");

        s = createSlider (false);
        s->setSliderStyle (Slider::LinearVertical);
        s->setVelocityBasedMode (true);
        s->setSkewFactor (0.5);
        s->setTextBoxStyle (Slider::TextBoxAbove, true, 100, 20);
        s->setBounds (sliderArea.removeFromLeft (70));
        s->setTextValueSuffix (" rels");

        sliderArea.removeFromLeft (20);
        auto horizonalSliderArea = sliderArea.removeFromLeft (180);

        s = createSlider (true);
        s->setSliderStyle (Slider::LinearHorizontal);
        s->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
        s->setBounds (horizonalSliderArea.removeFromTop (20));

        s = createSlider (false);
        s->setSliderStyle (Slider::LinearHorizontal);
        s->setTextBoxStyle (Slider::NoTextBox, false, 0, 0);
        horizonalSliderArea.removeFromTop (20);
        s->setBounds (horizonalSliderArea.removeFromTop (20));
        s->setPopupDisplayEnabled (true, false, this);
        s->setTextValueSuffix (" nuns required to change a lightbulb");

        s = createSlider (false);
        s->setSliderStyle (Slider::LinearHorizontal);
        s->setTextBoxStyle (Slider::TextEntryBoxPosition::TextBoxAbove,
            false, 70, 20);
        horizonalSliderArea.removeFromTop (20);
        s->setBounds (horizonalSliderArea.removeFromTop (50));
        s->setPopupDisplayEnabled (true, false, this);

        s = createSlider (false);
        s->setSliderStyle (Slider::IncDecButtons);
        s->setTextBoxStyle (Slider::TextBoxLeft, false, 50, 20);
        horizonalSliderArea.removeFromTop (20);
        s->setBounds (horizonalSliderArea.removeFromTop (20));
        s->setIncDecButtonsMode (Slider::incDecButtonsDraggable_Vertical);

        s = createSlider (false);
        s->setSliderStyle (Slider::Rotary);
        s->setRotaryParameters (MathConstants<float>::pi * 1.2f,
             MathConstants<float>::pi * 2.8f, false);
        s->setTextBoxStyle (Slider::TextBoxRight, false, 70, 20);
        horizonalSliderArea.removeFromTop (15);
        s->setBounds (horizonalSliderArea.removeFromTop (70));
        s->setTextValueSuffix (" mm");

        s = createSlider (false);
        s->setSliderStyle (Slider::LinearBar);
        horizonalSliderArea.removeFromTop (10);
        s->setBounds (horizonalSliderArea.removeFromTop (30));
        s->setTextValueSuffix (" gallons");

        sliderArea.removeFromLeft (20);
        auto twoValueSliderArea = sliderArea.removeFromLeft (180);

        s = createSlider (false);
        s->setSliderStyle (Slider::TwoValueHorizontal);
        s->setBounds (twoValueSliderArea.removeFromTop (40));

        s = createSlider (false);
        s->setSliderStyle (Slider::ThreeValueHorizontal);
        s->setPopupDisplayEnabled (true, false, this);
        twoValueSliderArea.removeFromTop (10);
        s->setBounds (twoValueSliderArea.removeFromTop (40));

        s = createSlider (false);
        s->setSliderStyle (Slider::TwoValueVertical);
        twoValueSliderArea.removeFromLeft (30);
        s->setBounds (twoValueSliderArea.removeFromLeft (40));

        s = createSlider (false);
        s->setSliderStyle (Slider::ThreeValueVertical);
        s->setPopupDisplayEnabled (true, false, this);
        twoValueSliderArea.removeFromLeft (30);
        s->setBounds (twoValueSliderArea.removeFromLeft (40));

        s = createSlider (false);
        s->setSliderStyle (Slider::LinearBarVertical);
        s->setTextBoxStyle (Slider::NoTextBox, false, 0, 0);
        sliderArea.removeFromLeft (20);
        s->setBounds (sliderArea.removeFromLeft (20));
        s->setPopupDisplayEnabled (true, true, this);
        s->setTextValueSuffix (" mickles in a muckle");

        /* Here, we'll create a Value object, and tell a bunch of our
             sliders to use it as their
           value source. By telling them all to share the same Value,
                they'll stay in sync with
           each other.

           We could also optionally keep a copy of this Value elsewhere, and
```

```
                by changing it,
           cause all the sliders to automatically update.
        */
        Value sharedValue;
        sharedValue = Random::getSystemRandom().nextDouble() * 100;
        for (int i = 0; i < 8; ++i)
            sliders.getUnchecked (i)->getValueObject().referTo (sharedValue)
                 ;

        // ..and now we'll do the same for all our min/max slider values..
        Value sharedValueMin, sharedValueMax;
        sharedValueMin = Random::getSystemRandom().nextDouble() * 40.0;
        sharedValueMax = Random::getSystemRandom().nextDouble() * 40.0 +
             60.0;

        for (int i = 8; i <= 11; ++i)
        {
            auto* selectedSlider = sliders.getUnchecked(i);
            selectedSlider->setTextBoxStyle (Slider::NoTextBox, false, 0, 0)
                 ;
            selectedSlider->getMaxValueObject().referTo (sharedValueMax);
            selectedSlider->getMinValueObject().referTo (sharedValueMin);
        }

        hintLabel.setBounds (layoutArea);
        addAndMakeVisible (hintLabel);
    }

private:
    OwnedArray<Slider> sliders;
    Label hintLabel  { "hint", "Try right-clicking on a slider for an
         options menu. \n\n"
                               "Also, holding down CTRL while dragging will
                                      turn on a slider's velocity-sensitive
                                      mode" };

    Slider* createSlider (bool isSnapping)
    {
        auto* s = isSnapping ? new SnappingSlider()
                             : new Slider();

        sliders.add (s);
        addAndMakeVisible (s);
        s->setRange (0.0, 100.0, 0.1);
        s->setPopupMenuEnabled (true);
        s->setValue (Random::getSystemRandom().nextDouble() * 100.0,
             dontSendNotification);
        return s;
    }

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (SlidersPage)
};

//
    ========================================================================
    ======
struct ButtonsPage   : public Component
{
    ButtonsPage (bool isRunningComponentTransformDemo)
    {
        {
            auto* group = addToList (new GroupComponent ("group", "Radio
                  buttons"));
            group->setBounds (20, 20, 220, 140);
        }

        for (int i = 0; i < 4; ++i)
        {
            auto* tb = addToList (new ToggleButton ("Radio Button #" +
                  String (i + 1)));

            tb->setRadioGroupId (1234);
            tb->setBounds (45, 46 + i * 22, 180, 22);
            tb->setTooltip ("A set of mutually-exclusive radio buttons");

            if (i == 0)
                tb->setToggleState (true, dontSendNotification);
        }

        for (int i = 0; i < 4; ++i)
        {
            DrawablePath normal, over;

            Path p;
            p.addStar ({}, i + 5, 20.0f, 50.0f, -0.2f);
            normal.setPath (p);
            normal.setFill (Colours::lightblue);
            normal.setStrokeFill (Colours::black);
            normal.setStrokeThickness (4.0f);

            over.setPath (p);
            over.setFill (Colours::blue);
            over.setStrokeFill (Colours::black);
            over.setStrokeThickness (4.0f);

            auto* db = addToList (new DrawableButton (String (i + 5) + "
                  points", DrawableButton::ImageAboveTextLabel));
            db->setImages (&normal, &over, nullptr);
            db->setClickingTogglesState (true);
            db->setRadioGroupId (23456);

            int buttonSize = 50;
            db->setBounds (25 + i * buttonSize, 180, buttonSize, buttonSize)
                  ;

            if (i == 0)
                db->setToggleState (true, dontSendNotification);
        }

        for (int i = 0; i < 4; ++i)
        {
            auto* tb = addToList (new TextButton ("Button " + String (i + 1)
                  ));

            tb->setClickingTogglesState (true);
            tb->setRadioGroupId (34567);
            tb->setColour (TextButton::textColourOffId,  Colours::black);
            tb->setColour (TextButton::textColourOnId,   Colours::black);
            tb->setColour (TextButton::buttonColourId,   Colours::white);
            tb->setColour (TextButton::buttonOnColourId, Colours::blueviolet
                .brighter());

            tb->setBounds (20 + i * 55, 260, 55, 24);
            tb->setConnectedEdges (((i != 0) ? Button::ConnectedOnLeft : 0)
                                    | ((i != 3) ? Button::ConnectedOnRight :
                                           0));

            if (i == 0)
                tb->setToggleState (true, dontSendNotification);
        }

        {
            auto* colourChangeButton = new ColourChangeButton();
            components.add (colourChangeButton);
            addAndMakeVisible (colourChangeButton);
            colourChangeButton->setTopLeftPosition (20, 320);
        }

        {
            auto* hyperlink = addToList (new HyperlinkButton ("This is a
                HyperlinkButton",
                                                              { "http://
                                                         www.juce.com" }));
            hyperlink->setBounds (260, 20, 200, 24);
        }

        // create some drawables to use for our drawable buttons...
        DrawablePath normal, over;

        {
            Path p;
            p.addStar ({}, 5, 20.0f, 50.0f, 0.2f);
            normal.setPath (p);
            normal.setFill (getRandomDarkColour());
        }

        {
            Path p;
            p.addStar ({}, 9, 25.0f, 50.0f, 0.0f);
            over.setPath (p);
            over.setFill (getRandomBrightColour());
            over.setStrokeFill (getRandomDarkColour());
            over.setStrokeThickness (5.0f);
        }

        DrawableImage down;
        down.setImage (getImageFromAssets ("juce_icon.png"));
        down.setOverlayColour (Colours::black.withAlpha (0.3f));

        auto popupMessageCallback = [this, isRunningComponentTransformDemo]
        {
            if (auto* focused = Component::getCurrentlyFocusedComponent())
                showBubbleMessage (*focused,
                                   "This is a demo of the
                                          BubbleMessageComponent, which lets
                                          you pop up a message pointing "
                                   "at a component or somewhere on the
                                          screen.\n\n"
                                   "The message bubbles will disappear after
                                          a timeout period, or when the
```

```
                                        mouse is clicked.",
                                 this->bubbleMessage,
                                 isRunningComponentTransformDemo);
        };

        {
            // create an image-above-text button from these drawables..
            auto db = addToList (new DrawableButton ("Button 1",
                 DrawableButton::ImageAboveTextLabel));
            db->setImages (&normal, &over, &down);
            db->setBounds (260, 60, 80, 80);
            db->setTooltip ("This is a DrawableButton with a label");
            db->onClick = popupMessageCallback;
        }

        {
            // create an image-only button from these drawables..
            auto db = addToList (new DrawableButton ("Button 2",
                 DrawableButton::ImageFitted));
            db->setImages (&normal, &over, &down);
            db->setClickingTogglesState (true);
            db->setBounds (370, 60, 80, 80);
            db->setTooltip ("This is an image-only DrawableButton");
            db->onClick = popupMessageCallback;
        }

        {
            // create an image-on-button-shape button from the same
                 drawables..
            auto db = addToList (new DrawableButton ("Button 3",
                 DrawableButton::ImageOnButtonBackground));
            db->setImages (&normal, nullptr, nullptr);
            db->setBounds (260, 160, 110, 25);
            db->setTooltip ("This is a DrawableButton on a standard button
                 background");
            db->onClick = popupMessageCallback;
        }

        {
            auto db = addToList (new DrawableButton ("Button 4",
                 DrawableButton::ImageOnButtonBackground));
            db->setImages (&normal, &over, &down);
            db->setClickingTogglesState (true);
            db->setColour (DrawableButton::backgroundColourId,   Colours::
                 white);
            db->setColour (DrawableButton::backgroundOnColourId, Colours::
                 yellow);
            db->setBounds (400, 150, 50, 50);
            db->setTooltip ("This is a DrawableButton on a standard button
                 background");
            db->onClick = popupMessageCallback;
        }

        {
            auto sb = addToList (new ShapeButton ("ShapeButton",
                                                  getRandomDarkColour(),
                                                  getRandomDarkColour(),
                                                  getRandomDarkColour()));
            sb->setShape (getJUCELogoPath(), false, true, false);
            sb->setBounds (260, 220, 200, 120);
        }

        {
            auto ib = addToList (new ImageButton ("ImageButton"));

            auto juceImage = getImageFromAssets ("juce_icon.png");

            ib->setImages (true, true, true,
                           juceImage, 0.7f, Colours::transparentBlack,
                           juceImage, 1.0f, Colours::transparentBlack,
                           juceImage, 1.0f, getRandomBrightColour().
                                 withAlpha (0.8f),
                           0.5f);

            ib->setBounds (260, 350, 100, 100);
            ib->setTooltip ("ImageButton - showing alpha-channel hit-testing
                 and colour overlay when clicked");
        }
    }

private:
    OwnedArray<Component> components;
    std::unique_ptr<BubbleMessageComponent> bubbleMessage;

    // This little function avoids a bit of code-duplication by adding a
        component to
    // our list as well as calling addAndMakeVisible on it..
    template <typename ComponentType>
    ComponentType* addToList (ComponentType* newComp)
    {
        components.add (newComp);
        addAndMakeVisible (newComp);
        return newComp;
    }

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (ButtonsPage)
};


//
    ========================================================================
    ======
struct MiscPage   : public Component
{
    MiscPage()
    {
        addAndMakeVisible (textEditor1);
        textEditor1.setBounds (10, 25, 200, 24);
        textEditor1.setText ("Single-line text box");

        addAndMakeVisible (textEditor2);
        textEditor2.setBounds (10, 55, 200, 24);
        textEditor2.setText ("Password");

        addAndMakeVisible (comboBox);
        comboBox.setBounds (10, 85, 200, 24);
        comboBox.setEditableText (true);
        comboBox.setJustificationType (Justification::centred);

        for (int i = 1; i < 100; ++i)
```

```
            comboBox.addItem ("combo box item " + String (i), i);

        comboBox.setSelectedId (1);
    }

    void lookAndFeelChanged() override
    {
        textEditor1.applyFontToAllText (textEditor1.getFont());
        textEditor2.applyFontToAllText (textEditor2.getFont());
    }

    TextEditor textEditor1,
               textEditor2  { "Password", (juce_wchar) 0x2022 };

    ComboBox comboBox  { "Combo" };
};

//
    ========================================================================
    ======
class ToolbarDemoComp   : public Component,
                          private Slider::Listener
{
public:
    ToolbarDemoComp()
    {
        // Create and add the toolbar...
        addAndMakeVisible (toolbar);

        // And use our item factory to add a set of default icons to it...
        toolbar.addDefaultItems (factory);

        // Now we'll just create the other sliders and buttons on the demo
             page, which adjust
        // the toolbar's properties...
        addAndMakeVisible (infoLabel);
        infoLabel.setJustificationType (Justification::topLeft);
        infoLabel.setBounds (80, 80, 450, 100);
        infoLabel.setInterceptsMouseClicks (false, false);

        addAndMakeVisible (depthSlider);
        depthSlider.setRange (10.0, 200.0, 1.0);
        depthSlider.setValue (50, dontSendNotification);
        depthSlider.addListener (this);
        depthSlider.setBounds (80, 210, 300, 22);
        depthLabel.attachToComponent (&depthSlider, false);

        addAndMakeVisible (orientationButton);
        orientationButton.onClick = [this] { toolbar.setVertical (! toolbar.
             isVertical()); resized(); };
        orientationButton.changeWidthToFitText (22);
        orientationButton.setTopLeftPosition (depthSlider.getX(),
             depthSlider.getBottom() + 20);

        addAndMakeVisible (customiseButton);
        customiseButton.onClick = [this] { toolbar.showCustomisationDialog
             (factory); };
        customiseButton.changeWidthToFitText (22);
        customiseButton.setTopLeftPosition (orientationButton.getRight() +
             20, orientationButton.getY());
    }

    void resized() override
    {
        auto toolbarThickness = (int) depthSlider.getValue();

        if (toolbar.isVertical())
            toolbar.setBounds (getLocalBounds().removeFromLeft
                 (toolbarThickness));
        else
            toolbar.setBounds (getLocalBounds().removeFromTop
                 (toolbarThickness));
    }

    void sliderValueChanged (Slider*) override
    {
        resized();
    }

private:
    Toolbar toolbar;

    Slider depthSlider  { Slider::LinearHorizontal, Slider::TextBoxLeft };

    Label depthLabel  { {}, "Toolbar depth:" },
          infoLabel   { {}, "As well as showing off toolbars, this demo
               illustrates how to store "
                            "a set of SVG files in a Zip file, embed that in
                                   your application, and read "
                            "them back in at runtime.\n\nThe icon images
                                   here are taken from the open-source "
                            "Tango icon project."};

    TextButton orientationButton  { "Vertical/Horizontal" },
               customiseButton    { "Customise..." };

    //
        ====================================================================
        ==========
    class DemoToolbarItemFactory   : public ToolbarItemFactory
    {
    public:
        DemoToolbarItemFactory() {}

        //
             ===============================================================
             ===============
        // Each type of item a toolbar can contain must be given a unique
             ID. These
        // are the ones we'll use in this demo.
        enum DemoToolbarItemIds
        {
            doc_new         = 1,
            doc_open        = 2,
            doc_save        = 3,
            doc_saveAs      = 4,
            edit_copy       = 5,
            edit_cut        = 6,
            edit_paste      = 7,
            juceLogoButton  = 8,
            customComboBox   = 9
        };

        void getAllToolbarItemIds (Array<int>& ids) override
        {
            // This returns the complete list of all item IDs that are
                 allowed to
            // go in our toolbar. Any items you might want to add must be
                 listed here. The
            // order in which they are listed will be used by the toolbar
                 customisation panel.

            ids.add (doc_new);
            ids.add (doc_open);
            ids.add (doc_save);
            ids.add (doc_saveAs);
            ids.add (edit_copy);
            ids.add (edit_cut);
            ids.add (edit_paste);
            ids.add (juceLogoButton);
            ids.add (customComboBox);

            // If you're going to use separators, then they must also be
                 added explicitly
            // to the list.
            ids.add (separatorBarId);
            ids.add (spacerId);
            ids.add (flexibleSpacerId);
        }

        void getDefaultItemSet (Array<int>& ids) override
        {
            // This returns an ordered list of the set of items that make up
                 a
            // toolbar's default set. Not all items need to be on this list,
                 and
            // items can appear multiple times (e.g. the separators used
                 here).
            ids.add (doc_new);
            ids.add (doc_open);
            ids.add (doc_save);
            ids.add (doc_saveAs);
            ids.add (spacerId);
            ids.add (separatorBarId);
            ids.add (edit_copy);
            ids.add (edit_cut);
            ids.add (edit_paste);
            ids.add (separatorBarId);
            ids.add (flexibleSpacerId);
            ids.add (customComboBox);
            ids.add (flexibleSpacerId);
            ids.add (separatorBarId);
            ids.add (juceLogoButton);
        }

        ToolbarItemComponent* createItem (int itemId) override
        {
            switch (itemId)
            {
                case doc_new:           return createButtonFromZipFileSVG
                (itemId, "new",      "document-new.svg");
            case doc_open:           return createButtonFromZipFileSVG
                (itemId, "open",     "document-open.svg");
            case doc_save:           return createButtonFromZipFileSVG
                (itemId, "save",     "document-save.svg");
            case doc_saveAs:         return createButtonFromZipFileSVG
                (itemId, "save as", "document-save-as.svg");
            case edit_copy:          return createButtonFromZipFileSVG
                (itemId, "copy",     "edit-copy.svg");
            case edit_cut:           return createButtonFromZipFileSVG
                (itemId, "cut",      "edit-cut.svg");
            case edit_paste:         return createButtonFromZipFileSVG
                (itemId, "paste",    "edit-paste.svg");
            case juceLogoButton:
            {
                auto* drawable = new DrawableImage();
                drawable->setImage (getImageFromAssets ("juce_icon.png")
                      );
                return new ToolbarButton (itemId, "juce!", drawable,
                      nullptr);
            }
            case customComboBox:     return new CustomToolbarComboBox
                (itemId);
            default:                 break;
        }

        return nullptr;
    }

private:
    StringArray iconNames;
    OwnedArray<Drawable> iconsFromZipFile;

    // This is a little utility to create a button with one of the SVG
         images in
    // our embedded ZIP file "icons.zip"
    ToolbarButton* createButtonFromZipFileSVG (const int itemId, const
         String& text, const String& filename)
    {
        if (iconsFromZipFile.size() == 0)
        {
            // If we've not already done so, load all the images from
                 the zip file..
            ZipFile icons (createAssetInputStream ("icons.zip"), true);

            for (int i = 0; i < icons.getNumEntries(); ++i)
            {
                std::unique_ptr<InputStream> svgFileStream (icons.
                      createStreamForEntry (i));

                if (svgFileStream.get() != nullptr)
                {
                    iconNames.add (icons.getEntry (i)->filename);
                    iconsFromZipFile.add (Drawable::
                          createFromImageDataStream (*svgFileStream));
                }
            }
        }

        auto* image = iconsFromZipFile[iconNames.indexOf (filename)]->
```

```
                createCopy();
            return new ToolbarButton (itemId, text, image, nullptr);
        }

        // Demonstrates how to put a custom component into a toolbar - this
             one contains
        // a ComboBox.
        class CustomToolbarComboBox : public ToolbarItemComponent
        {
        public:
            CustomToolbarComboBox (const int toolbarItemId)
                : ToolbarItemComponent (toolbarItemId, "Custom Toolbar Item"
                     , false)
            {
                addAndMakeVisible (comboBox);

                for (int i = 1; i < 20; ++i)
                    comboBox.addItem ("Toolbar ComboBox item " + String (i),
                          i);

                comboBox.setSelectedId (1);
                comboBox.setEditableText (true);
            }

            bool getToolbarItemSizes (int /*toolbarDepth*/, bool isVertical,
                                      int& preferredSize, int& minSize, int&
                                             maxSize) override
            {
                if (isVertical)
                    return false;

                preferredSize = 250;
                minSize = 80;
                maxSize = 300;
                return true;
            }

            void paintButtonArea (Graphics&, int, int, bool, bool) override
            {
            }

            void contentAreaChanged (const Rectangle<int>& newArea) override
            {
                comboBox.setSize (newArea.getWidth() - 2,
                                  jmin (newArea.getHeight() - 2, 22));

                comboBox.setCentrePosition (newArea.getCentreX(), newArea.
                     getCentreY());
            }

        private:
            ComboBox comboBox  { "demo toolbar combo box" };
        };
    };

    DemoToolbarItemFactory factory;
};


//
```

```
        ==========================================================================
        ======
    /**
        This class shows how to implement a TableListBoxModel to show in a
            TableListBox.
    */
    class TableDemoComponent    : public Component,
                                  public TableListBoxModel
    {
    public:
        TableDemoComponent()
        {
            // Load some data from an embedded XML file..
            loadData();

            // Create our table component and add it to this component..
            addAndMakeVisible (table);
            table.setModel (this);

            // give it a border
            table.setColour (ListBox::outlineColourId, Colours::grey);
            table.setOutlineThickness (1);

            // Add some columns to the table header, based on the column list in
                our database..
            forEachXmlChildElement (*columnList, columnXml)
            {
                table.getHeader().addColumn (columnXml->getStringAttribute
                    ("name"),
                                             columnXml->getIntAttribute
                                                 ("columnId"),
                                             columnXml->getIntAttribute ("width"
                                                 ),
                                             50, 400,
                                             TableHeaderComponent::defaultFlags)
                                                 ;
            }

            // we could now change some initial settings..
            table.getHeader().setSortColumnId (1, true); // sort forwards by the
                ID column
            table.getHeader().setColumnVisible (7, false); // hide the "length"
                column until the user shows it

            // un-comment this line to have a go of stretch-to-fit mode
            // table.getHeader().setStretchToFitActive (true);

            table.setMultipleSelectionEnabled (true);
        }

        // This is overloaded from TableListBoxModel, and must return the total
            number of rows in our table
        int getNumRows() override
        {
            return numRows;
        }

        // This is overloaded from TableListBoxModel, and should fill in the
            background of the whole row
        void paintRowBackground (Graphics& g, int rowNumber, int /*width*/,
             int /*height*/, bool rowIsSelected) override
        {
            auto alternateColour = getLookAndFeel().findColour (ListBox::
                 backgroundColourId)
                                                  .interpolatedWith
                                                       (getLookAndFeel().
                                                       findColour (ListBox::
                                                       textColourId), 0.03f);
            if (rowIsSelected)
                g.fillAll (Colours::lightblue);
            else if (rowNumber % 2)
                g.fillAll (alternateColour);
        }

        // This is overloaded from TableListBoxModel, and must paint any cells
             that aren't using custom
        // components.
        void paintCell (Graphics& g, int rowNumber, int columnId,
                        int width, int height, bool /*rowIsSelected*/) override
        {
            g.setColour (getLookAndFeel().findColour (ListBox::textColourId));
            g.setFont (font);

            if (auto* rowElement = dataList->getChildElement (rowNumber))
            {
                auto text = rowElement->getStringAttribute
                     (getAttributeNameForColumnId (columnId));

                g.drawText (text, 2, 0, width - 4, height, Justification::
                     centredLeft, true);
            }

            g.setColour (getLookAndFeel().findColour (ListBox::
                 backgroundColourId));
            g.fillRect (width - 1, 0, 1, height);
        }

        // This is overloaded from TableListBoxModel, and tells us that the user
             has clicked a table header
        // to change the sort order.
        void sortOrderChanged (int newSortColumnId, bool isForwards) override
        {
            if (newSortColumnId != 0)
            {
                DemoDataSorter sorter (getAttributeNameForColumnId
                     (newSortColumnId), isForwards);
                dataList->sortChildElements (sorter);

                table.updateContent();
            }
        }

        // This is overloaded from TableListBoxModel, and must update any custom
             components that we're using
        Component* refreshComponentForCell (int rowNumber, int columnId, bool /
             *isRowSelected*/,
                                            Component* existingComponentToUpdate
                                                    ) override
        {
            if (columnId == 1 || columnId == 7) // The ID and Length columns do
```

```
             not have a custom component
        {
            jassert (existingComponentToUpdate == nullptr);
            return nullptr;
        }

        if (columnId == 5) // For the ratings column, we return the custom
             combobox component
        {
            auto* ratingsBox = static_cast<RatingColumnCustomComponent*>
                 (existingComponentToUpdate);

            // If an existing component is being passed-in for updating,
                 we'll re-use it, but
            // if not, we'll have to create one.
            if (ratingsBox == nullptr)
                ratingsBox = new RatingColumnCustomComponent (*this);

            ratingsBox->setRowAndColumn (rowNumber, columnId);
            return ratingsBox;
        }

        // The other columns are editable text columns, for which we use the
             custom Label component
        auto* textLabel = static_cast<EditableTextCustomComponent*>
             (existingComponentToUpdate);

        // same as above...
        if (textLabel == nullptr)
            textLabel = new EditableTextCustomComponent (*this);

        textLabel->setRowAndColumn (rowNumber, columnId);
        return textLabel;
    }

    // This is overloaded from TableListBoxModel, and should choose the best
         width for the specified
    // column.
    int getColumnAutoSizeWidth (int columnId) override
    {
        if (columnId == 5)
            return 100; // (this is the ratings column, containing a custom
                 combobox component)

        int widest = 32;

        // find the widest bit of text in this column..
        for (int i = getNumRows(); --i >= 0;)
        {
            if (auto* rowElement = dataList->getChildElement (i))
            {
                auto text = rowElement->getStringAttribute
                     (getAttributeNameForColumnId (columnId));

                widest = jmax (widest, font.getStringWidth (text));
            }
        }

        return widest + 8;
    }
    // A couple of quick methods to set and get cell values when the user
         changes them
    int getRating (const int rowNumber) const
    {
        return dataList->getChildElement (rowNumber)->getIntAttribute
             ("Rating");
    }

    void setRating (const int rowNumber, const int newRating)
    {
        dataList->getChildElement (rowNumber)->setAttribute ("Rating",
             newRating);
    }

    String getText (const int columnNumber, const int rowNumber) const
    {
        return dataList->getChildElement (rowNumber)->getStringAttribute (
             getAttributeNameForColumnId(columnNumber));
    }

    void setText (const int columnNumber, const int rowNumber, const String&
         newText)
    {
        auto columnName = table.getHeader().getColumnName (columnNumber);
        dataList->getChildElement (rowNumber)->setAttribute (columnName,
             newText);
    }

    //
         ====================================================================
         ==========
    void resized() override
    {
        // position our table with a gap around its edge
        table.setBoundsInset (BorderSize<int> (8));
    }


private:
    TableListBox table;     // the table component itself
    Font font  { 14.0f };

    std::unique_ptr<XmlElement> demoData;  // This is the XML document
         loaded from the embedded file "demo table data.xml"
    XmlElement* columnList = nullptr;     // A pointer to the sub-node of
         demoData that contains the list of columns
    XmlElement* dataList   = nullptr;     // A pointer to the sub-node of
         demoData that contains the list of data rows
    int numRows;                          // The number of rows of data
         we've got

    //
         ====================================================================
         ==========
    // This is a custom Label component, which we use for the table's
         editable text columns.
    class EditableTextCustomComponent  : public Label
    {
    public:
        EditableTextCustomComponent (TableDemoComponent& td)  : owner (td)
        {
            // double click to edit the label text; single click handled
                below
            setEditable (false, true, false);
        }

        void mouseDown (const MouseEvent& event) override
        {
            // single click on the label should simply select the row
            owner.table.selectRowsBasedOnModifierKeys (row, event.mods,
                false);

            Label::mouseDown (event);
        }

        void textWasEdited() override
        {
            owner.setText (columnId, row, getText());
        }

        // Our demo code will call this when we may need to update our
            contents
        void setRowAndColumn (const int newRow, const int newColumn)
        {
            row = newRow;
            columnId = newColumn;
            setText (owner.getText(columnId, row), dontSendNotification);
        }

        void paint (Graphics& g) override
        {
            auto& lf = getLookAndFeel();
            if (! dynamic_cast<LookAndFeel_V4*> (&lf))
                lf.setColour (textColourId, Colours::black);

            Label::paint (g);
        }

    private:
        TableDemoComponent& owner;
        int row, columnId;
        Colour textColour;
    };

    //
        ====================================================================
        ==========
    // This is a custom component containing a combo box, which we're going
        to put inside
    // our table's "rating" column.
    class RatingColumnCustomComponent    : public Component
    {
    public:
        RatingColumnCustomComponent (TableDemoComponent& td)  : owner (td)
        {
            // just put a combo box inside this component
            addAndMakeVisible (comboBox);
            comboBox.addItem ("fab",        1);
            comboBox.addItem ("groovy",     2);
            comboBox.addItem ("hep",        3);
            comboBox.addItem ("mad for it", 4);
            comboBox.addItem ("neat",       5);
            comboBox.addItem ("swingin",    6);
            comboBox.addItem ("wild",       7);

            comboBox.onChange = [this] { owner.setRating (row, comboBox.
                 getSelectedId()); };
            comboBox.setWantsKeyboardFocus (false);
        }

        void resized() override
        {
            comboBox.setBoundsInset (BorderSize<int> (2));
        }

        // Our demo code will call this when we may need to update our
             contents
        void setRowAndColumn (int newRow, int newColumn)
        {
            row = newRow;
            columnId = newColumn;
            comboBox.setSelectedId (owner.getRating (row),
                 dontSendNotification);
        }

    private:
        TableDemoComponent& owner;
        ComboBox comboBox;
        int row, columnId;
    };

    //
         ====================================================================
         ==========
    // A comparator used to sort our data when the user clicks a column
         header
    class DemoDataSorter
    {
    public:
        DemoDataSorter (const String& attributeToSortBy, bool forwards)
            : attributeToSort (attributeToSortBy),
              direction (forwards ? 1 : -1)
        {
        }

        int compareElements (XmlElement* first, XmlElement* second) const
        {
            auto result = first->getStringAttribute (attributeToSort)
                                  .compareNatural (second->getStringAttribute
                                         (attributeToSort));

            if (result == 0)
                result = first->getStringAttribute ("ID")
                                  .compareNatural (second->getStringAttribute
                                         ("ID"));

            return direction * result;
        }
    private:
        String attributeToSort;
        int direction;
    };

    //
        ====================================================================
        ==========
    // this loads the embedded database XML file into memory
    void loadData()
    {
        demoData = parseXML (loadEntireAssetIntoString ("demo table
            data.xml"));

        dataList   = demoData->getChildByName ("DATA");
        columnList = demoData->getChildByName ("COLUMNS");

        numRows = dataList->getNumChildElements();
    }

    // (a utility method to search our XML for the attribute that matches a
        column ID)
    String getAttributeNameForColumnId (const int columnId) const
    {
        forEachXmlChildElement (*columnList, columnXml)
        {
            if (columnXml->getIntAttribute ("columnId") == columnId)
                return columnXml->getStringAttribute ("name");
        }

        return {};
    }

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (TableDemoComponent)
};

//
    ========================================================================
    ======
class DragAndDropDemo  : public Component,
                         public DragAndDropContainer
{
public:
    DragAndDropDemo()
    {
        setName ("Drag-and-Drop");

        sourceListBox.setModel (&sourceModel);
        sourceListBox.setMultipleSelectionEnabled (true);

        addAndMakeVisible (sourceListBox);
        addAndMakeVisible (target);
    }

    void resized() override
    {
        auto r = getLocalBounds().reduced (8);

        sourceListBox.setBounds (r.withSize (250, 180));
        target       .setBounds (r.removeFromBottom (150).removeFromRight
             (250));
    }

private:
    //
        ====================================================================
        ==========
    struct SourceItemListboxContents  : public ListBoxModel
    {
        // The following methods implement the necessary virtual functions
             from ListBoxModel,
        // telling the listbox how many rows there are, painting them, etc.
        int getNumRows() override
        {
            return 30;
        }

        void paintListBoxItem (int rowNumber, Graphics& g,
                               int width, int height, bool rowIsSelected)
                                      override
        {
            if (rowIsSelected)
                g.fillAll (Colours::lightblue);

            g.setColour (LookAndFeel::getDefaultLookAndFeel().findColour
                  (Label::textColourId));
            g.setFont (height * 0.7f);

            g.drawText ("Draggable Thing #" + String (rowNumber + 1),
                        5, 0, width, height,
                        Justification::centredLeft, true);
        }

        var getDragSourceDescription (const SparseSet<int>& selectedRows)
             override
        {
            // for our drag description, we'll just make a comma-separated
                 list of the selected row
            // numbers - this will be picked up by the drag target and
                 displayed in its box.
            StringArray rows;

            for (int i = 0; i < selectedRows.size(); ++i)
                rows.add (String (selectedRows[i] + 1));

            return rows.joinIntoString (", ");
        }
    };

    //
        ====================================================================
        ==========
    // and this is a component that can have things dropped onto it..
    class DragAndDropDemoTarget : public Component,
                                  public DragAndDropTarget,
                                  public FileDragAndDropTarget,
                                  public TextDragAndDropTarget
    {
    public:
        DragAndDropDemoTarget()    {}
        void paint (Graphics& g) override
        {
            g.fillAll (Colours::green.withAlpha (0.2f));

            // draw a red line around the comp if the user's currently
                dragging something over it..
            if (somethingIsBeingDraggedOver)
            {
                g.setColour (Colours::red);
                g.drawRect (getLocalBounds(), 3);
            }

            g.setColour (getLookAndFeel().findColour (Label::textColourId));
            g.setFont (14.0f);
            g.drawFittedText (message, getLocalBounds().reduced (10, 0),
                Justification::centred, 4);
        }

        //
            ===============================================================
            ===============
        // These methods implement the DragAndDropTarget interface, and
            allow our component
        // to accept drag-and-drop of objects from other JUCE components..

        bool isInterestedInDragSource (const SourceDetails& /
            *dragSourceDetails*/) override
        {
            // normally you'd check the sourceDescription value to see if
                it's the
            // sort of object that you're interested in before returning
                true, but for
            // the demo, we'll say yes to anything..
            return true;
        }

        void itemDragEnter (const SourceDetails& /*dragSourceDetails*/)
            override
        {
            somethingIsBeingDraggedOver = true;
            repaint();
        }

        void itemDragMove (const SourceDetails& /*dragSourceDetails*/)
            override
        {
        }

        void itemDragExit (const SourceDetails& /*dragSourceDetails*/)
            override
        {
            somethingIsBeingDraggedOver = false;
            repaint();
        }

        void itemDropped (const SourceDetails& dragSourceDetails) override
        {
            message = "Items dropped: " + dragSourceDetails.description.
                toString();
            somethingIsBeingDraggedOver = false;
            repaint();
        }

        //
            ===============================================================
            ===============
        // These methods implement the FileDragAndDropTarget interface, and
            allow our component
        // to accept drag-and-drop of files..

        bool isInterestedInFileDrag (const StringArray& /*files*/) override
        {
            // normally you'd check these files to see if they're something
                that you're
            // interested in before returning true, but for the demo, we'll
                say yes to anything..
            return true;
        }

        void fileDragEnter (const StringArray& /*files*/, int /*x*/, int /
            *y*/) override
        {
            somethingIsBeingDraggedOver = true;
            repaint();
        }

        void fileDragMove (const StringArray& /*files*/, int /*x*/, int /
            *y*/) override
        {
        }

        void fileDragExit (const StringArray& /*files*/) override
        {
            somethingIsBeingDraggedOver = false;
            repaint();
        }

        void filesDropped (const StringArray& files, int /*x*/, int /*y*/)
            override
        {
            message = "Files dropped: " + files.joinIntoString ("\n");

            somethingIsBeingDraggedOver = false;
            repaint();
        }

        //
            ===============================================================
            ===============
        // These methods implement the TextDragAndDropTarget interface, and
            allow our component
        // to accept drag-and-drop of text..

        bool isInterestedInTextDrag (const String& /*text*/) override
        {
            return true;
        }
```

```
        void textDragEnter (const String& /*text*/, int /*x*/, int /*y*/)
             override
        {
            somethingIsBeingDraggedOver = true;
            repaint();
        }

        void textDragMove (const String& /*text*/, int /*x*/, int /*y*/)
             override
        {
        }

        void textDragExit (const String& /*text*/) override
        {
            somethingIsBeingDraggedOver = false;
            repaint();
        }

        void textDropped (const String& text, int /*x*/, int /*y*/) override
        {
            message = "Text dropped:\n" + text;

            somethingIsBeingDraggedOver = false;
            repaint();
        }

    private:
        String message  { "Drag-and-drop some rows from the top-left box
             onto this component!\n\n"
                          "You can also drag-and-drop files and text from
                                other apps"};
        bool somethingIsBeingDraggedOver = false;
    };

    //
         ====================================================================
         ==========
    ListBox sourceListBox  { "D+D source", nullptr };
    SourceItemListboxContents sourceModel;
    DragAndDropDemoTarget target;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (DragAndDropDemo)
};

//
     ========================================================================
     ======
struct DemoTabbedComponent  : public TabbedComponent
{
    DemoTabbedComponent (bool isRunningComponenTransformsDemo)
        : TabbedComponent (TabbedButtonBar::TabsAtTop)
    {
        auto colour = findColour (ResizableWindow::backgroundColourId);

        addTab ("Buttons",     colour, new ButtonsPage
             (isRunningComponenTransformsDemo), true);
        addTab ("Sliders",     colour, new SlidersPage(),
             true);
        addTab ("Toolbars",    colour, new ToolbarDemoComp(),
             true);
        addTab ("Misc",        colour, new MiscPage(),
             true);
        addTab ("Tables",      colour, new TableDemoComponent(),
             true);
        addTab ("Drag & Drop", colour, new DragAndDropDemo(),
             true);

        getTabbedButtonBar().getTabButton (5)->setExtraComponent (new
             CustomTabButton (isRunningComponenTransformsDemo),

                                                                 TabBarButton::afterText)
                                                                 ;
    }

    // This is a small star button that is put inside one of the tabs. You
         can
    // use this technique to create things like "close tab" buttons, etc.
    class CustomTabButton  : public Component
    {
    public:
        CustomTabButton (bool isRunningComponenTransformsDemo)
            : runningComponenTransformsDemo (isRunningComponenTransformsDemo
                 )
        {
            setSize (20, 20);
        }

        void paint (Graphics& g) override
        {
            Path star;
            star.addStar ({}, 7, 1.0f, 2.0f);

            g.setColour (Colours::green);
            g.fillPath (star, star.getTransformToScaleToFit (getLocalBounds
                 ().reduced (2).toFloat(), true));
        }

        void mouseDown (const MouseEvent&) override
        {
            showBubbleMessage (*this,
                               "This is a custom tab component\n"
                               "\n"
                               "You can use these to implement things like
                                      close-buttons "
                               "or status displays for your tabs.",
                               bubbleMessage,
                               runningComponenTransformsDemo);
        }
    private:
        bool runningComponenTransformsDemo;
        std::unique_ptr<BubbleMessageComponent> bubbleMessage;
    };

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (DemoTabbedComponent)
};

//
    ========================================================================
    ======
struct WidgetsDemo   : public Component
```

```
{
    WidgetsDemo (bool isRunningComponenTransformsDemo = false)
        : tabs (isRunningComponenTransformsDemo)
    {
        setOpaque (true);
        addAndMakeVisible (tabs);

        setSize (700, 500);
    }

    void paint (Graphics& g) override
    {
        g.fillAll (Colours::lightgrey);
    }

    void resized() override
    {
        tabs.setBounds (getLocalBounds().reduced (4));
    }

    DemoTabbedComponent tabs;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (WidgetsDemo)
};

//
    ===================================================================
    ======
void showBubbleMessage (Component& targetComponent, const String& textToShow
    ,
                        std::unique_ptr<BubbleMessageComponent>& bmc,
                        bool isRunningComponentTransformDemo)
{
    bmc.reset (new BubbleMessageComponent());

    if (isRunningComponentTransformDemo)
    {
        targetComponent.findParentComponentOfClass<WidgetsDemo>()->
             addChildComponent (bmc.get());
    }
    else if (Desktop::canUseSemiTransparentWindows())
    {
        bmc->setAlwaysOnTop (true);
        bmc->addToDesktop (0);
    }
    else
    {
        targetComponent.getTopLevelComponent()->addChildComponent (bmc.get()
             );
    }

    AttributedString text (textToShow);
    text.setJustification (Justification::centred);
    text.setColour (targetComponent.findColour (TextButton::textColourOffId)
         );

    bmc->showAt (&targetComponent, text, 2000, true, false);
}
```