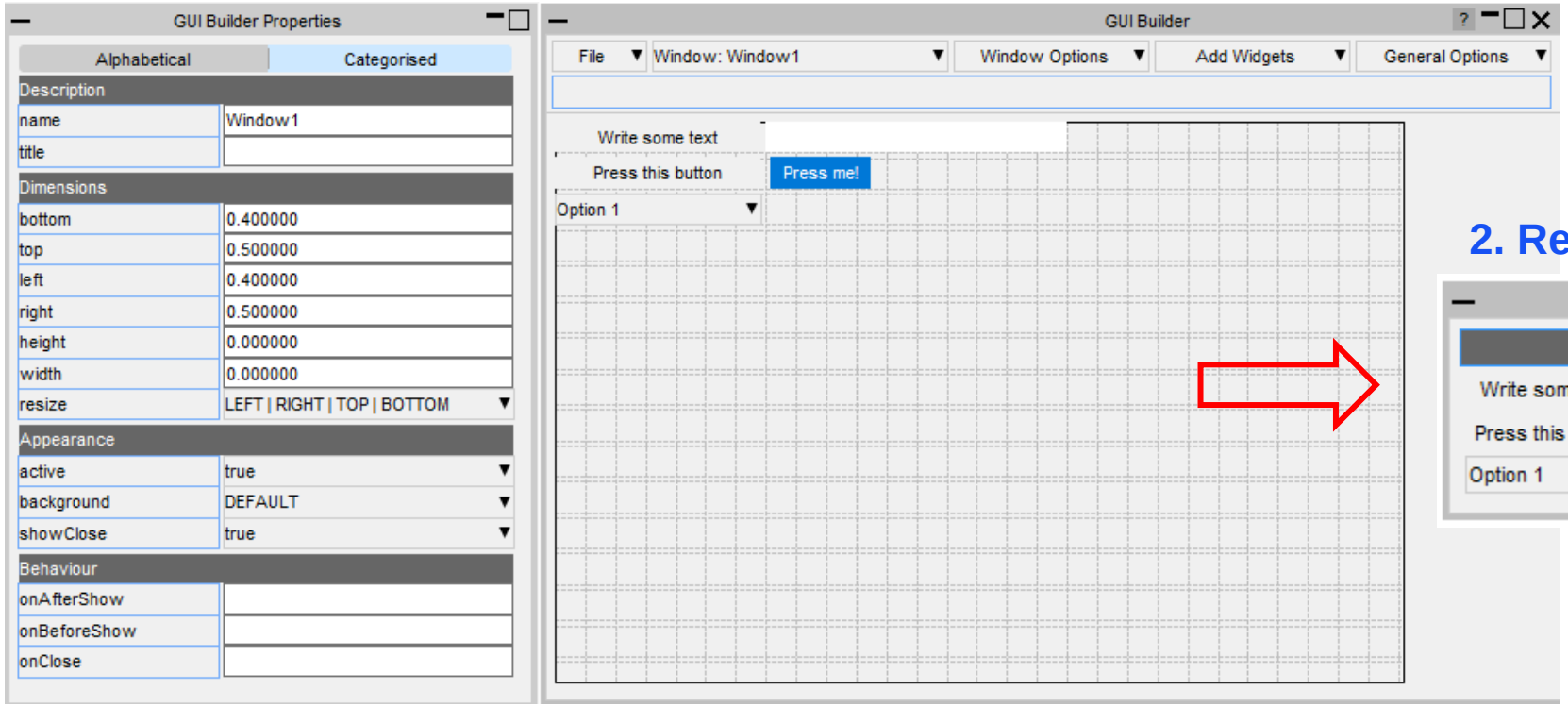
JavaScript GUI Builder

About

PRIMER

The JavaScript GUI Builder allows users to interactively design and build Graphical User Interfaces to use in a JavaScript, rather than having to write code.

1. Design and Save your GUI to a file

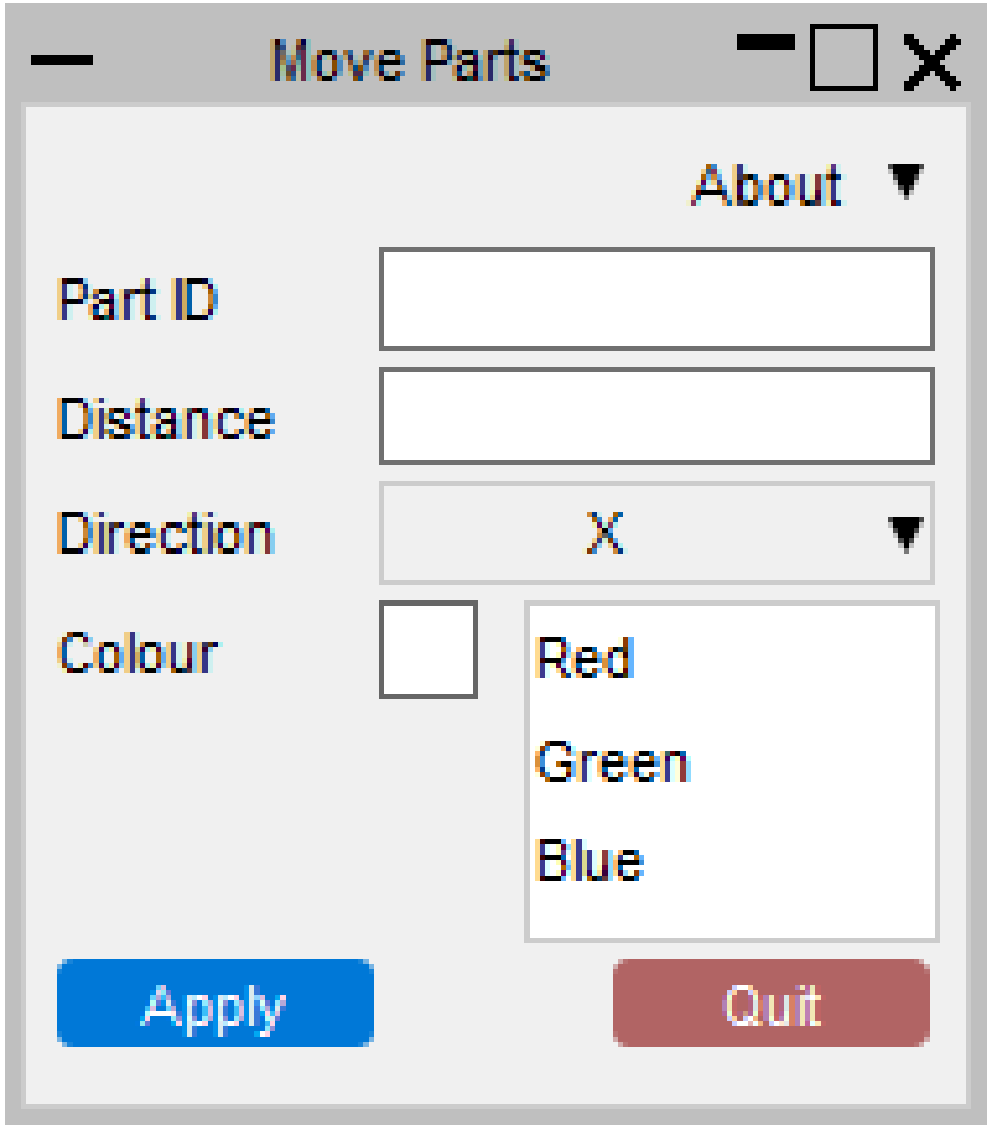


2. Read the file in your script

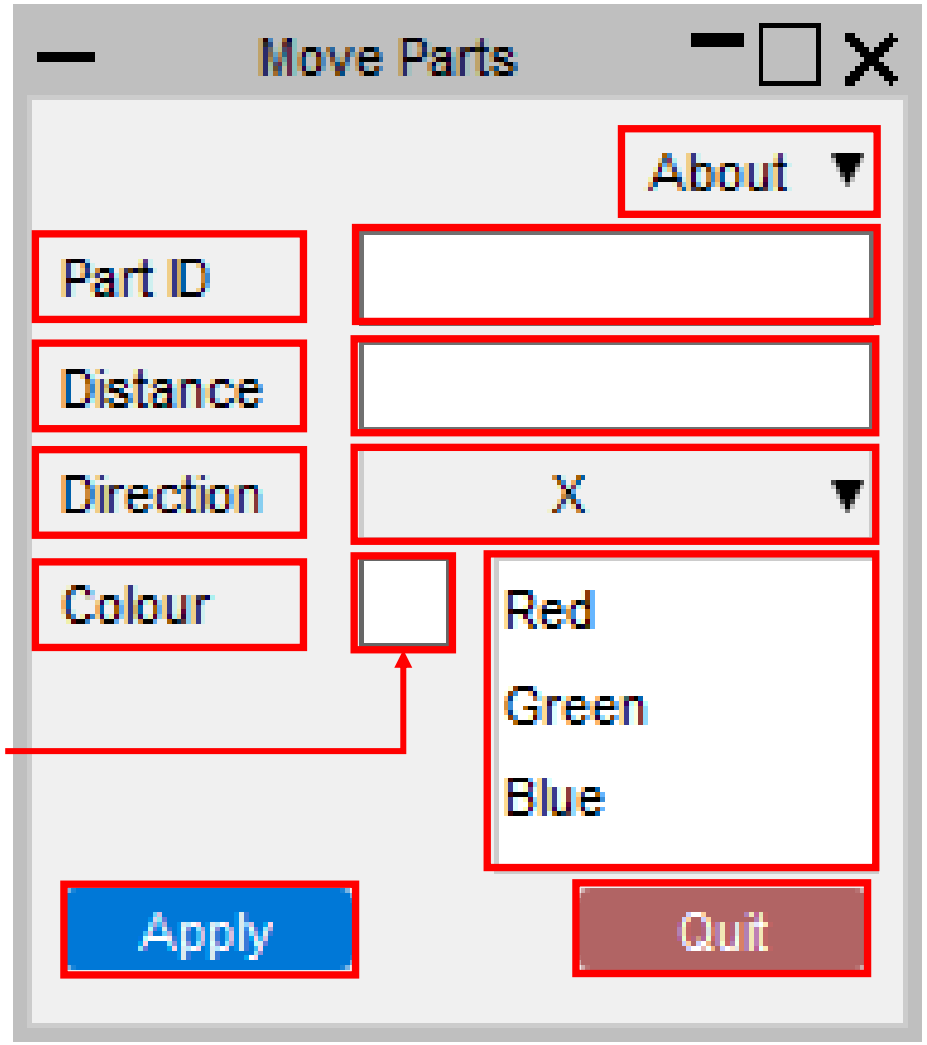
Outline

This tutorial will take you through the process of using the GUI Builder to create a GUI.

The example worked through will create a utility to move a part in PRIMER and optionally colour it.



Outline



Label Widgets

Checkbox Widget to toggle the colour

Button Widget to move the part

Label Widget with a PopupWindow

Textbox Widget to specify the part to move

Textbox Widget to specify the distance to move

Combobox Widget to specify the direction to move

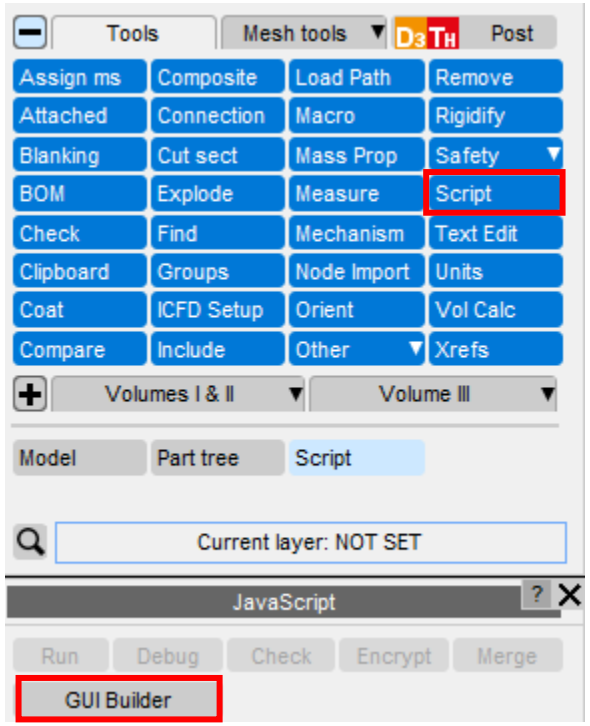
Listbox Widget to specify the colour

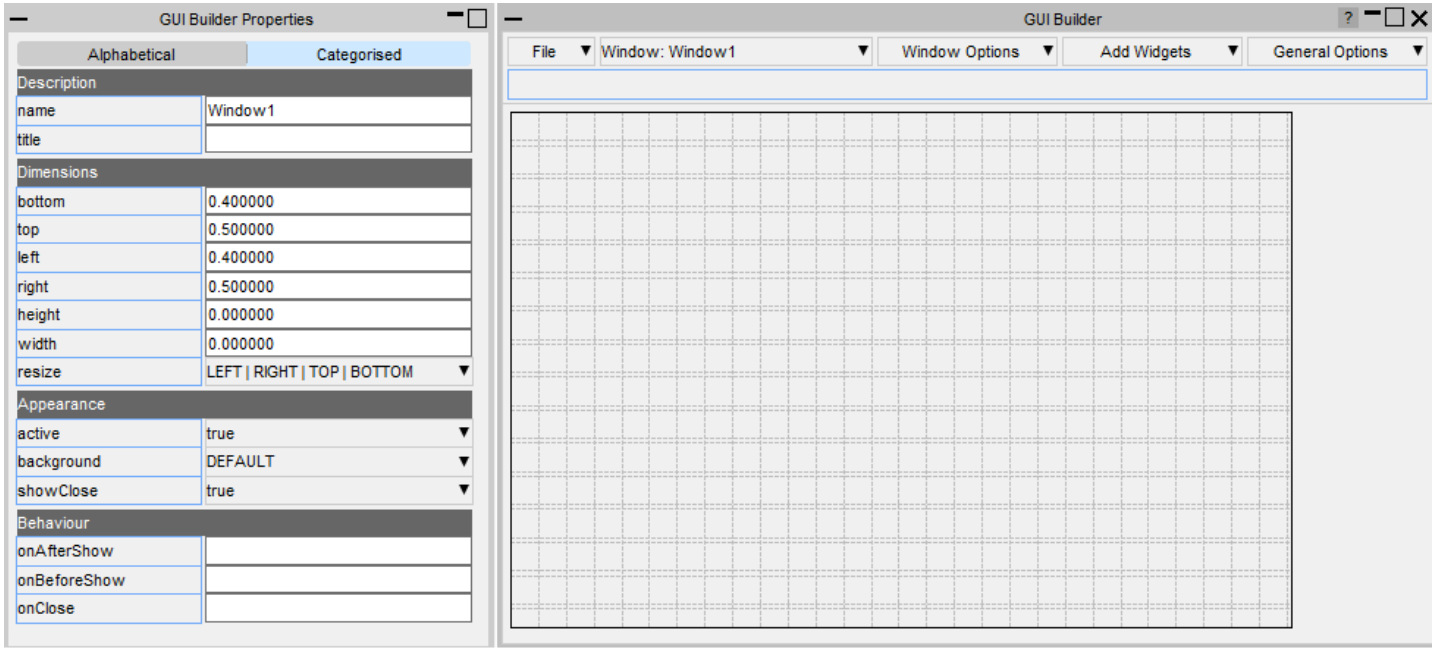
Button Widget to close the window

Open the GUI Builder

Open the GUI Builder

To open the GUI Builder in PRIMER go to *Script->GUI Builder*



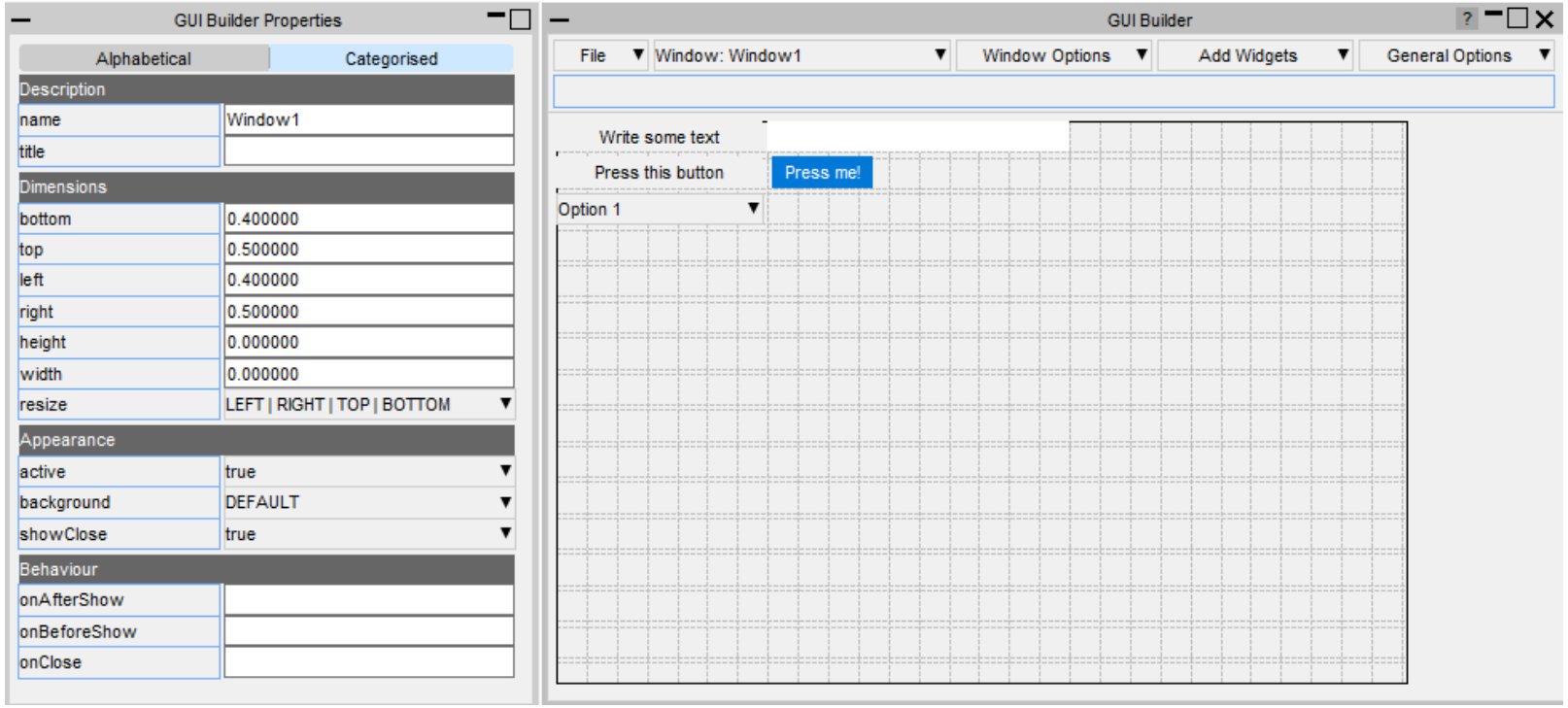


GUI Builder Overview

The GUI Builder is split into two windows, the Properties Window and the Design Window

Properties Window

The properties of widgets and windows are set here.



Design Window

Widgets are added, positioned and resized here.

Name the Window

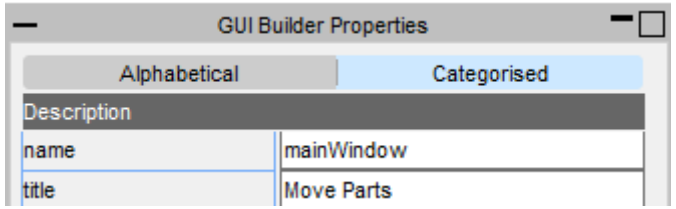
Name the Window

Every Window, Widget and WidgetItem object created in the GUI Builder has a *name* property. This is used to access each object when the GUI is read into a script.

The GUI Builder automatically generates names, but it is good practice to give them sensible names to make it easier to work with them in your scripts.

For this example set the Window name to ‘mainWindow’.

You can also give it a title ‘Move Parts’.

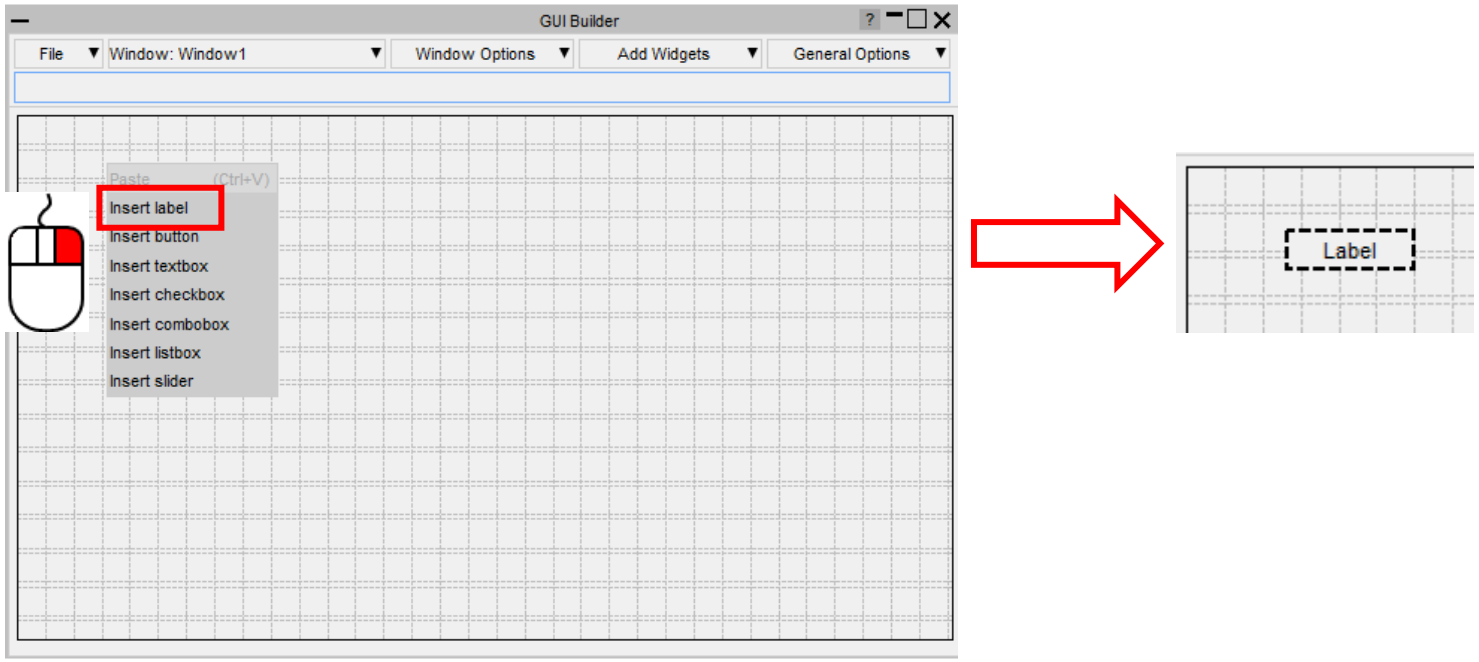


Create Widgets

Add Label Widgets

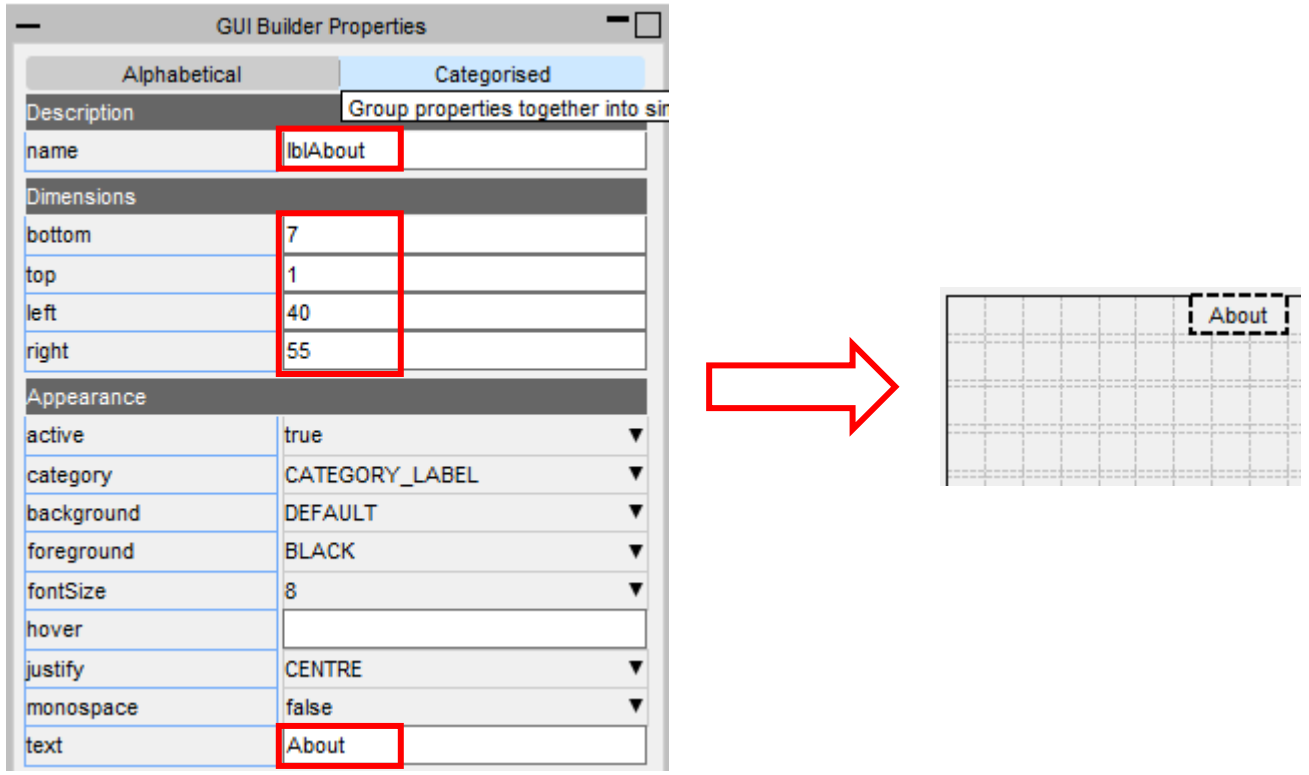
First we'll add the Label Widgets.

Right-click anywhere in the Design Window and select 'Insert label'



Add Label Widgets

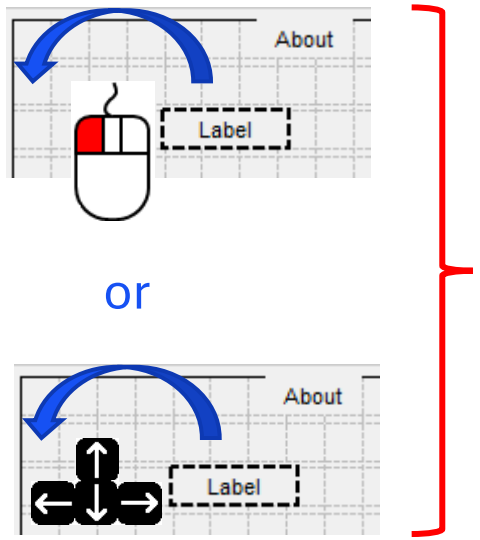
In the Properties Window set the highlighted properties (make sure the Label widget is selected by left-clicking on it):



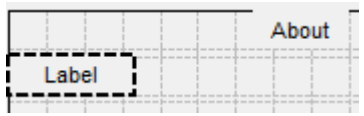
Add Label Widgets

Right-click in the Design Window and create another Label.

Move the Label, either by left-clicking on the Label and dragging, or by using arrow keys, so it's positioned as below.



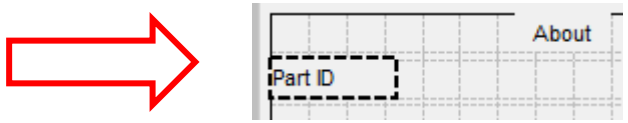
Dimensions	
bottom	14
top	8
left	1
right	21



Add Label Widgets

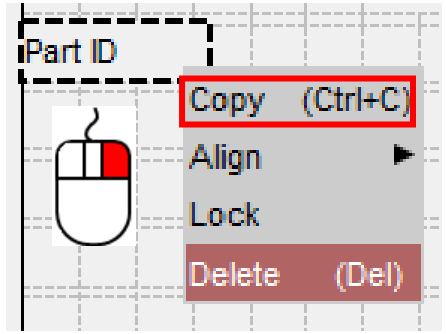
In the Properties Window set the highlighted properties:

Description	
name	lblPartId
Dimensions	
bottom	14
top	8
left	1
right	21
Appearance	
active	true
category	CATEGORY_LABEL
background	DEFAULT
foreground	BLACK
fontSize	8
hover	
justify	LEFT
monospace	false
text	Part ID



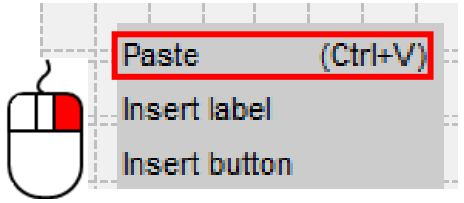
Add Label Widgets

Copy the ‘Part Id’ Label, either by right-clicking on it and selecting ‘Copy’, or left-clicking on it and pressing ‘Ctrl-C’.



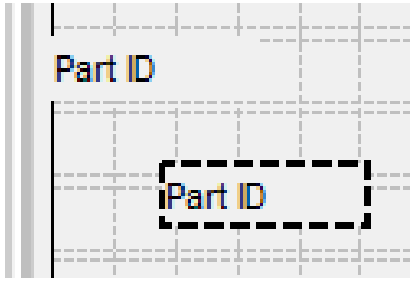
or

Create a new Label either by right-clicking on the Design Window and selecting ‘Paste’, or by pressing ‘Ctrl-V’



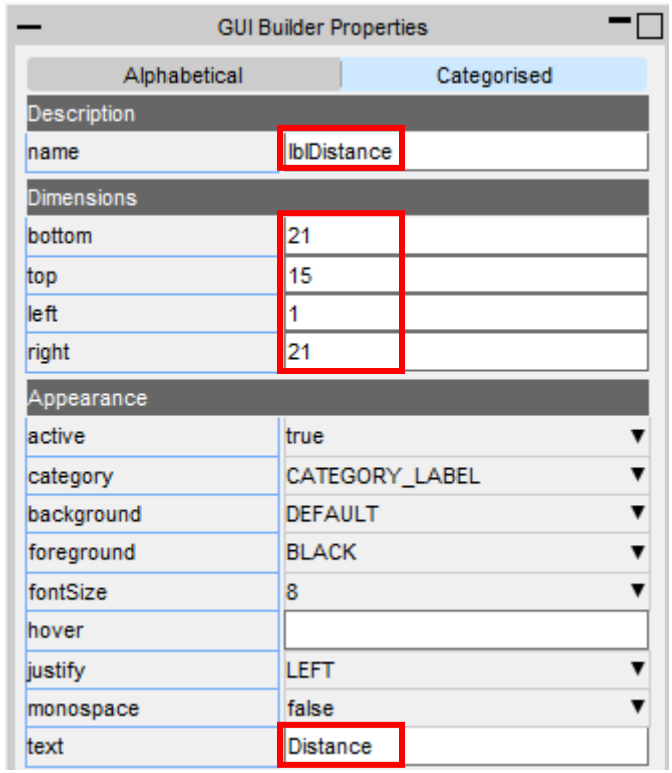
or

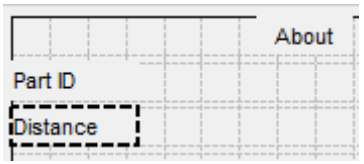
The new Label will have the same properties as the copied Label, except for the name and dimensions



Add Label Widgets

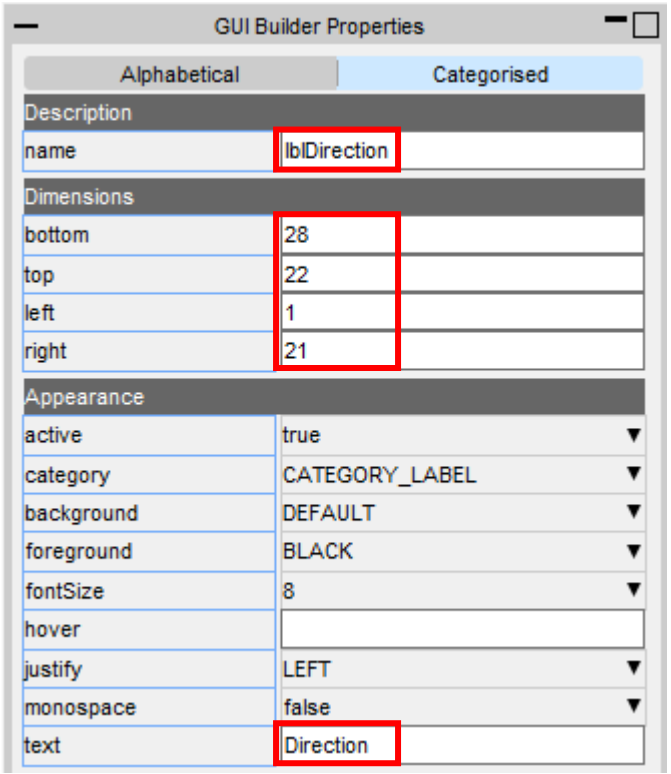
In the Properties Window set the highlighted properties:

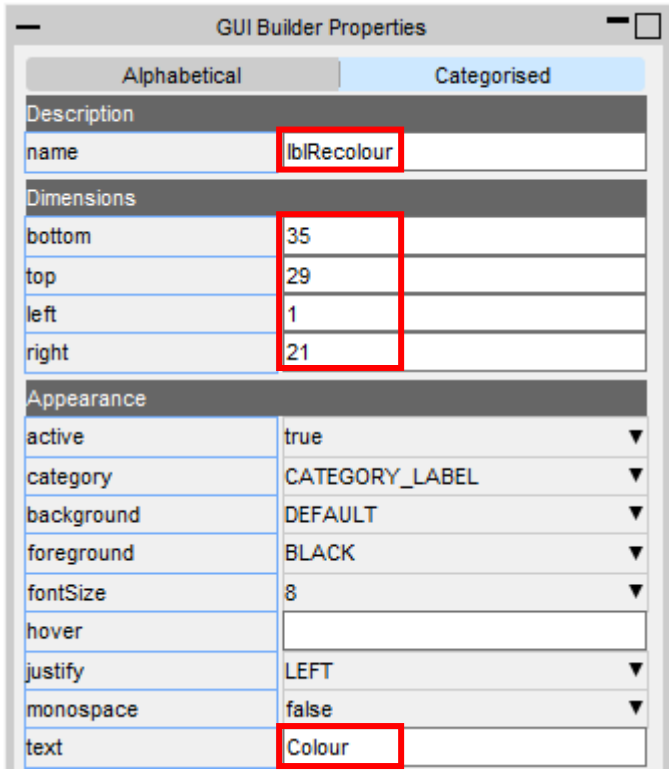


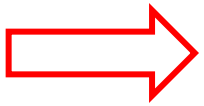


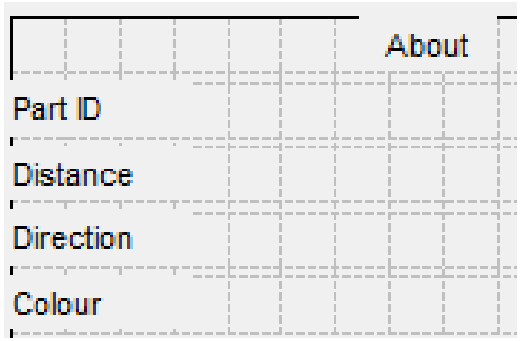
Add Label Widgets

Repeat the previous steps to create two more labels with the highlighted properties:





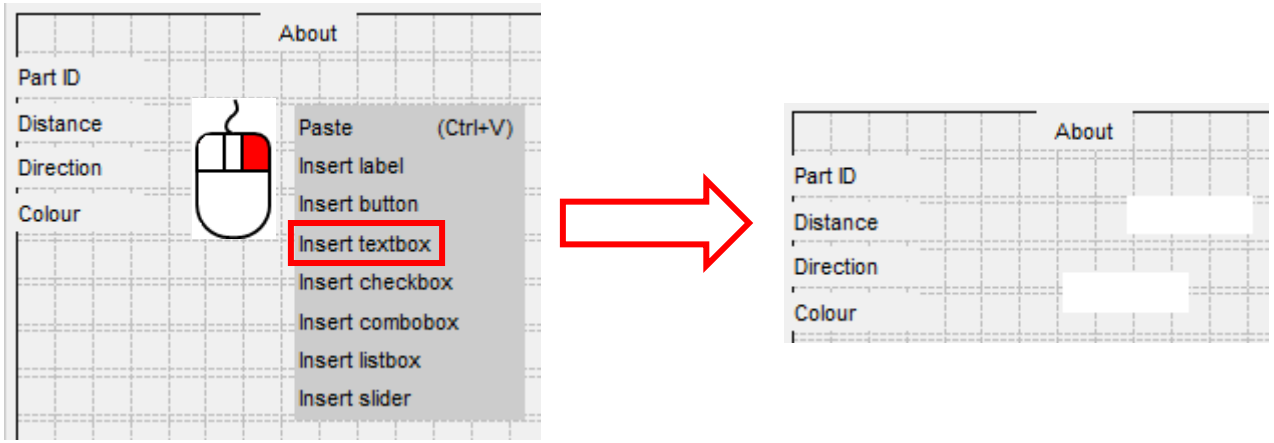




Add Textbox Widgets

Now we'll add the Textbox Widgets.

Right-click anywhere in the Design Window and select 'Insert textbox'. Do this twice to create two Textbox Widgets.



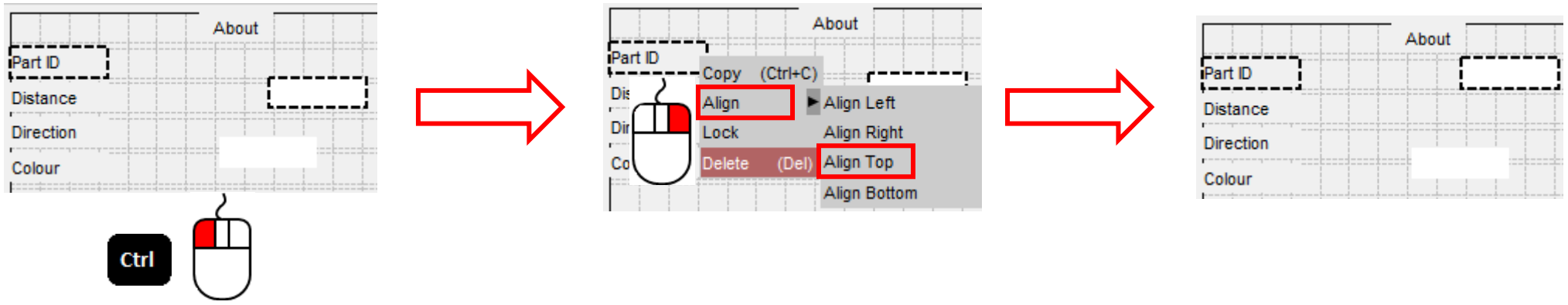
Add Textbox Widgets

To position them, we'll use the Align feature.

Holding the Ctrl key, left-click on one of the Textboxes and the 'Part ID' Label to select them both.

Then right-click on the 'Part ID' Label and select Align->Align Top.

The Textbox should move so the top aligns with the Label.



Add Textbox Widgets

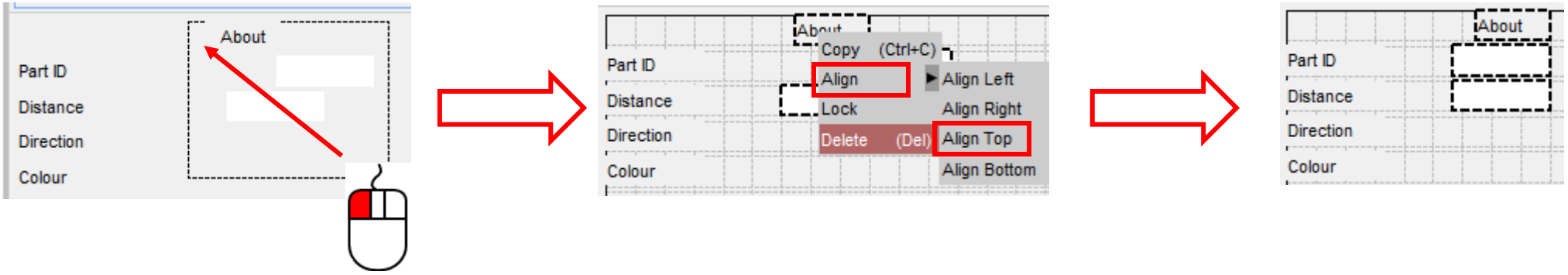
Do the same for the second Textbox, aligning the top with the ‘Distance’ Label.

The GUI should now look like this:

Add Textbox Widgets

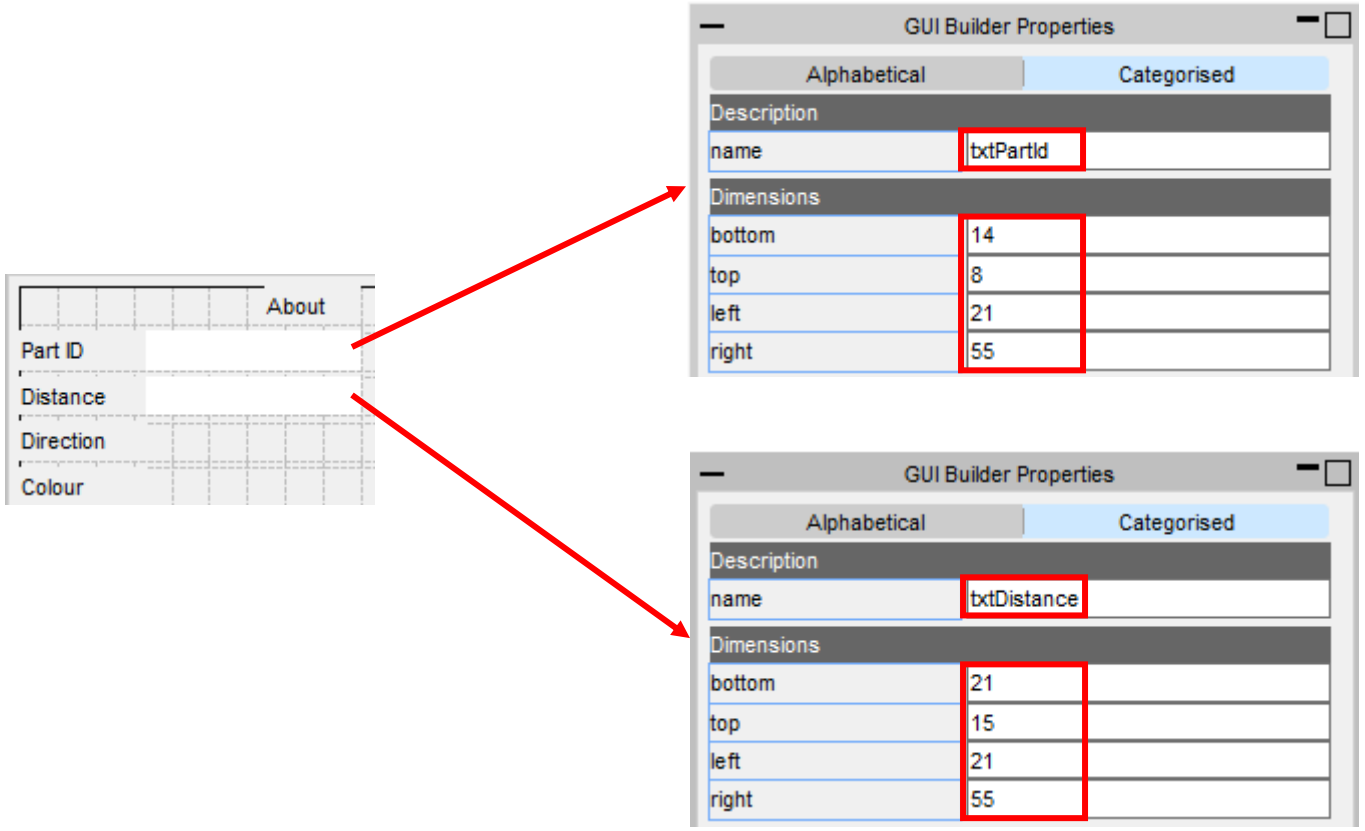
Select the ‘About’ Label and the two Textboxes by left-clicking and dragging a box around them.

Then right-click on the ‘About’ Label and select Align->Align Right.



Add Textbox Widgets

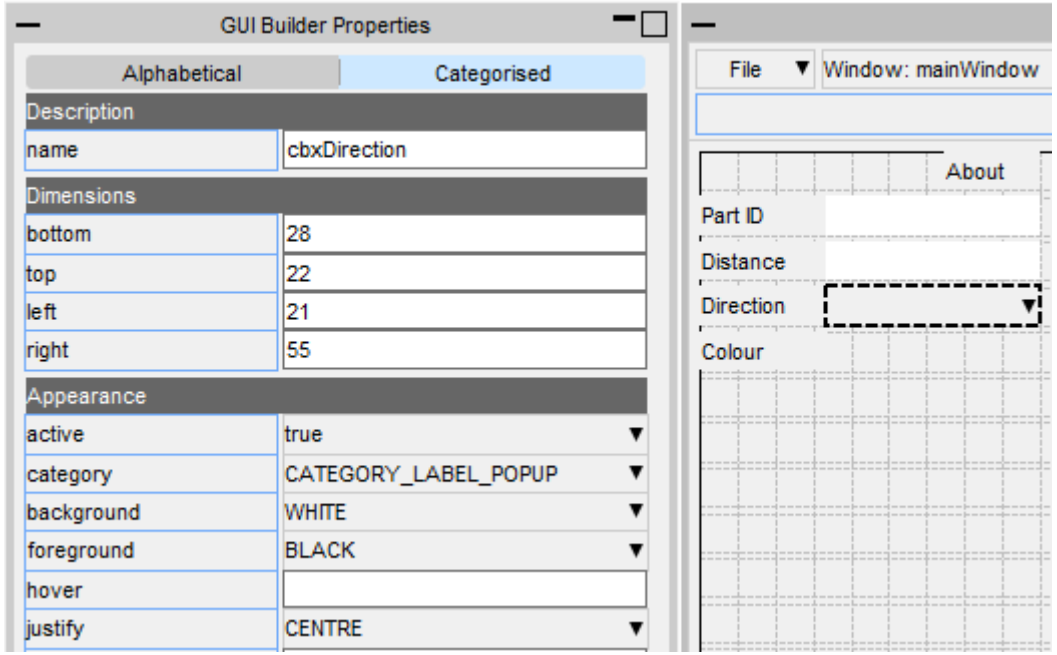
Set the properties of the two Textboxes



Add Combobox Widget

We'll now add the Combobox to select the direction to move the part.

Right-click anywhere in the Design Window and select 'Insert combobox'. Then position it as shown below, name it 'cbxDirection' and set the justify property to CENTRE.

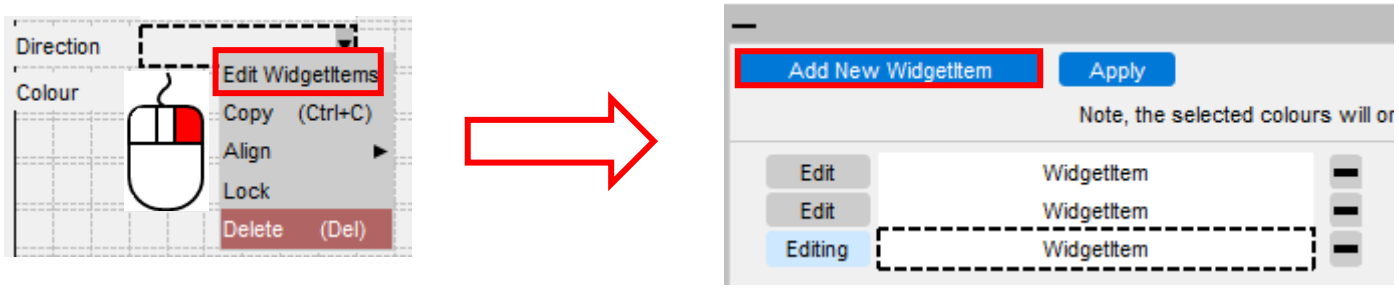


Add Combobox Widget

Right-click on the Combobox and select ‘Edit WidgetItems’.

The Design Window will update to allow you to add WidgetItems.

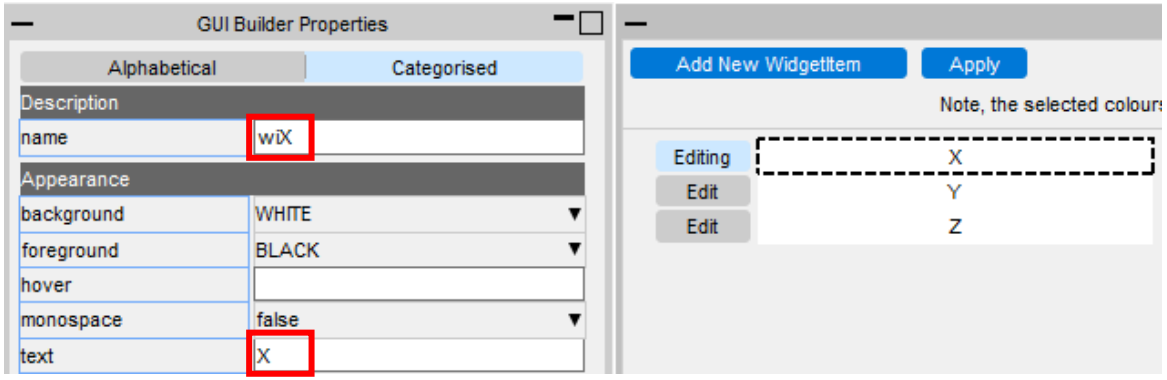
Press the ‘Add New WidgetItem’ button 3 times to create 3 WidgetItems

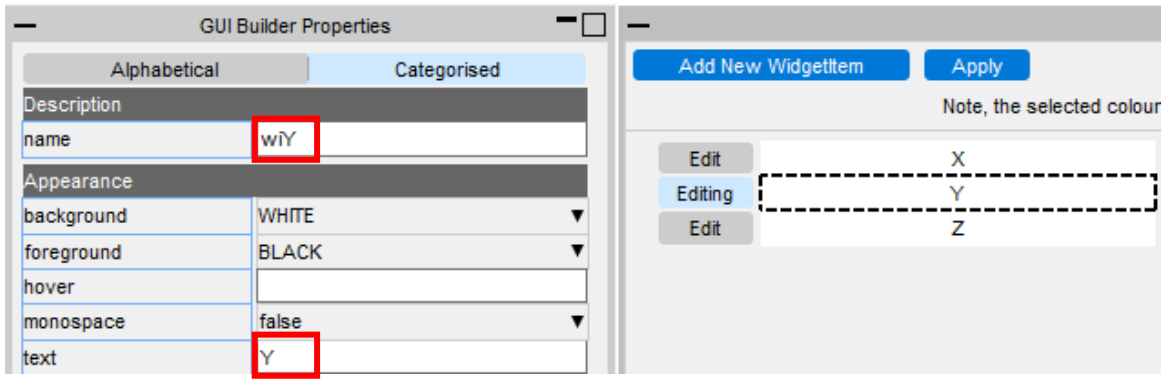


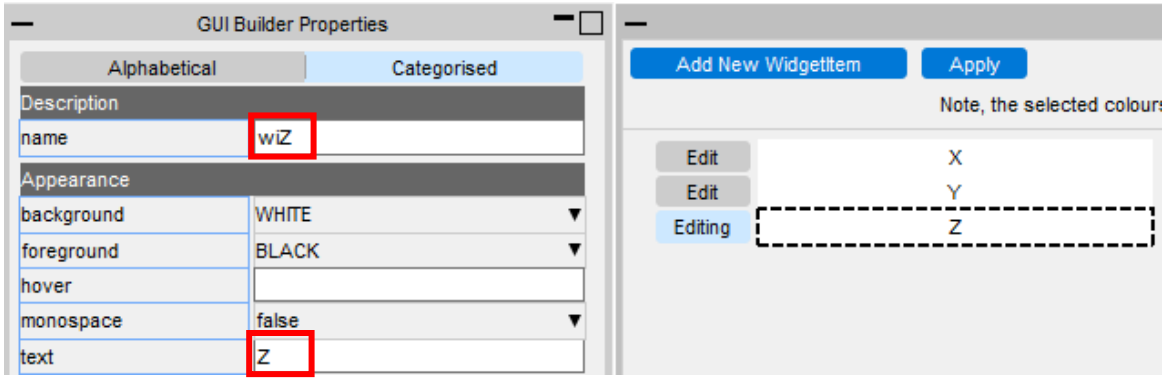
Add Combobox Widget

Select each WidgetItem in turn and set the properties as below.

Then press the Apply button to return to the Design Window.



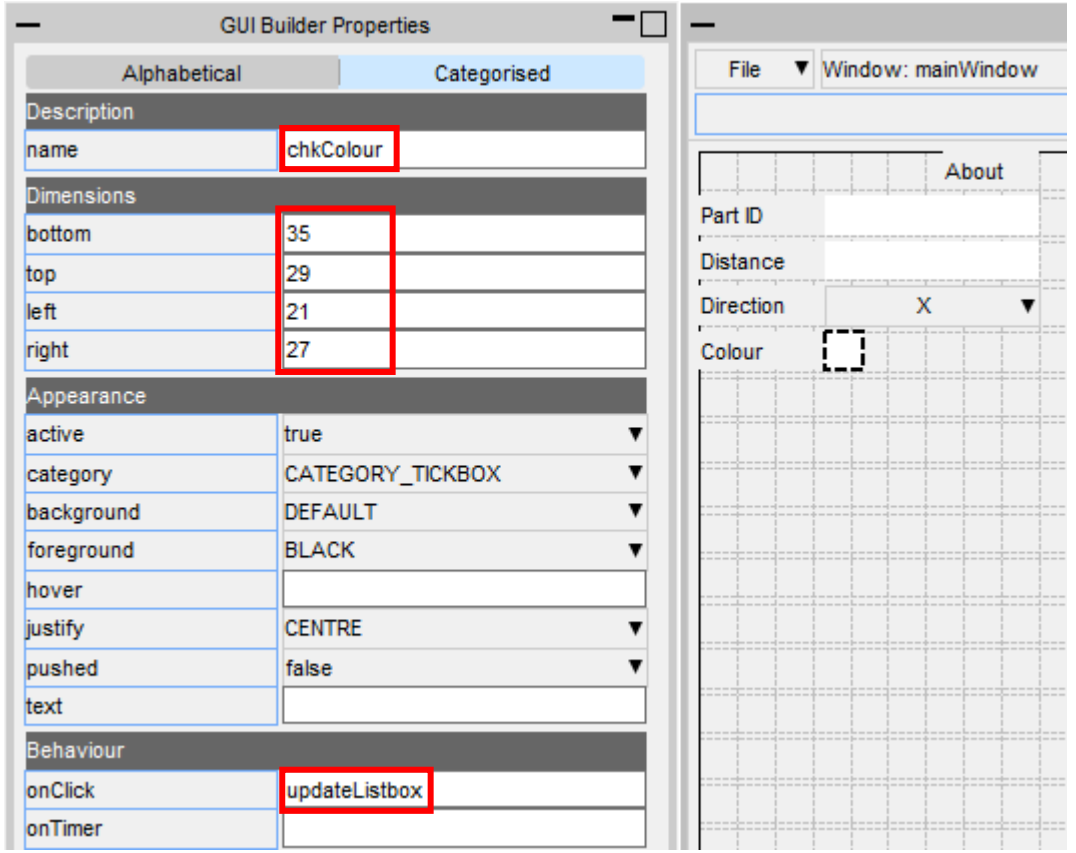




Add Checkbox Widget

Right-click anywhere in the Design Window and select ‘Insert checkbox’. Then position it as shown below and name it ‘chkColour’.

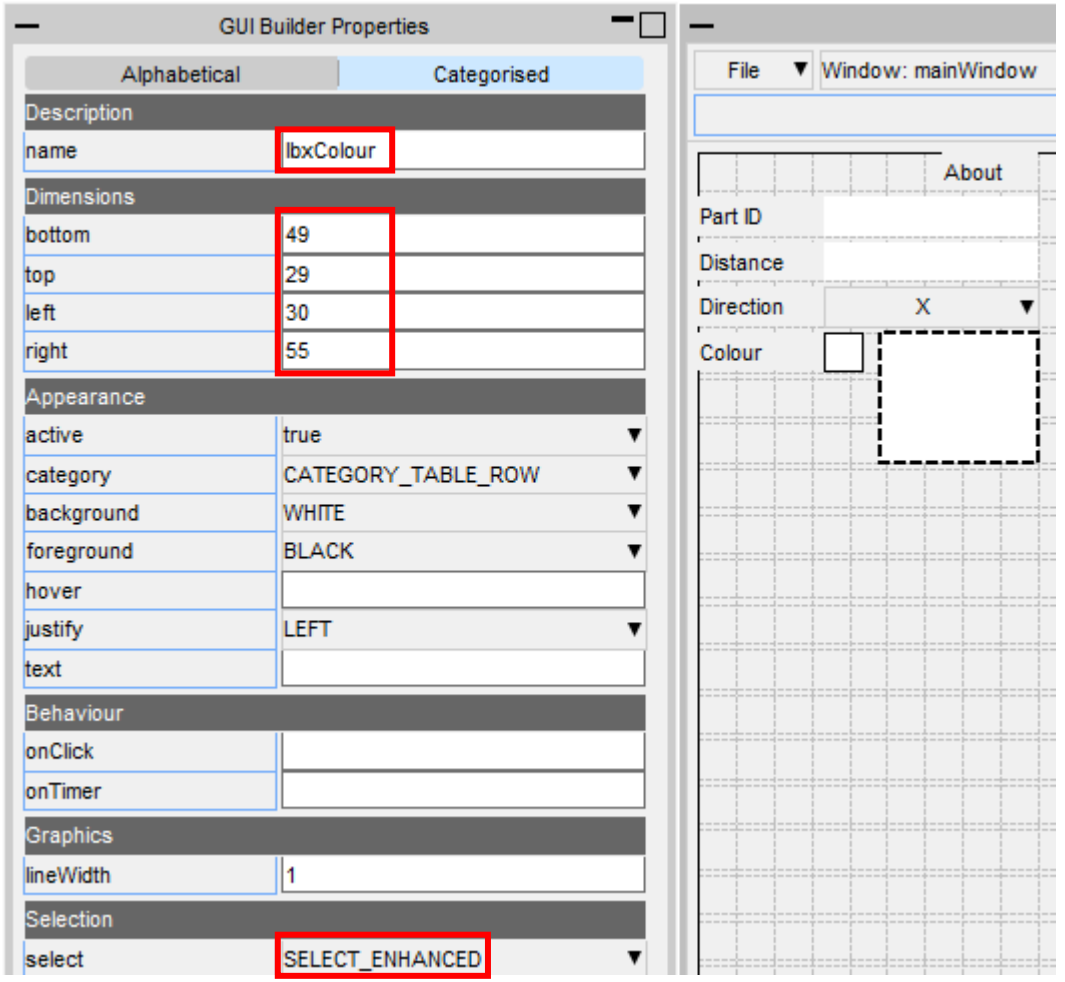
Set the onClick property to ‘updateListbox’. This is the name of a function that will be called when the user clicks on the Checkbox. We will have to write the contents of this function later once we’ve saved the GUI.



Add Listbox Widget

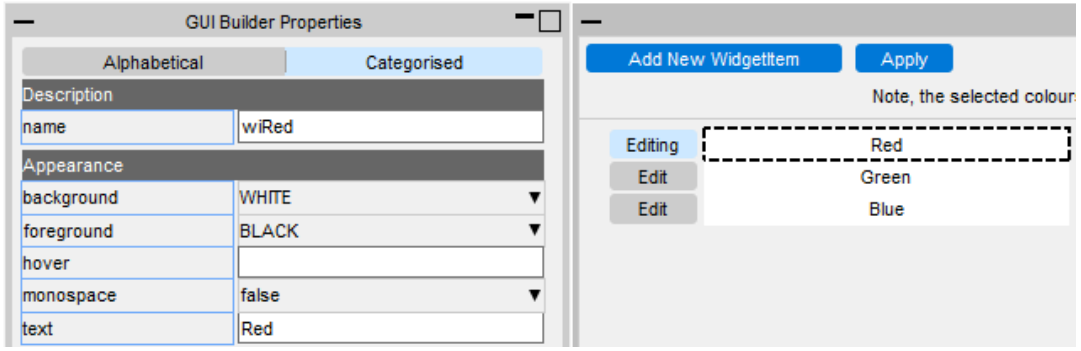
Right-click anywhere in the Design Window and select ‘Insert listbox’. Then position it as shown below and name it ‘lbxColour’.

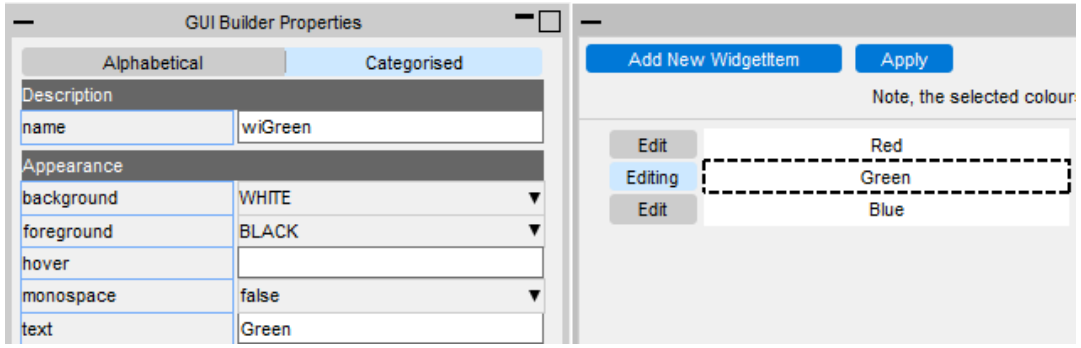
Set the select property to SELECT_SINGLE, so only one option can be selected at a time.

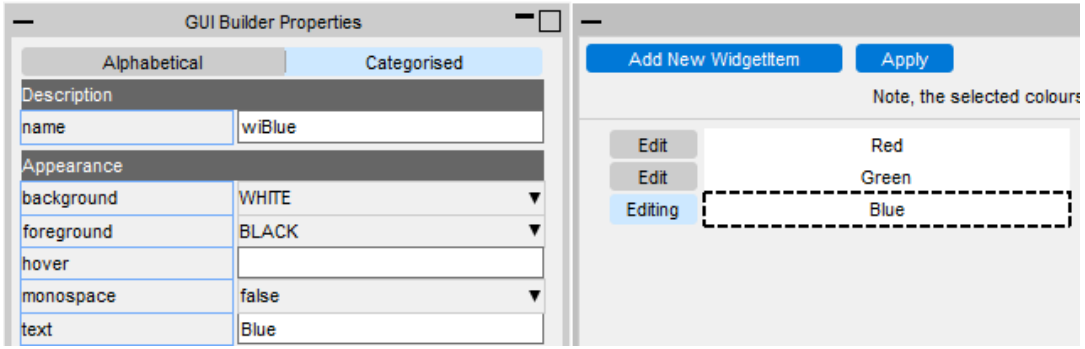


Add Listbox Widget

Right-click on the Listbox and select ‘Edit WidgetItems’. Add 3 WidgetItems and set the properties as below. Then press the Apply button to return to the Design Window.



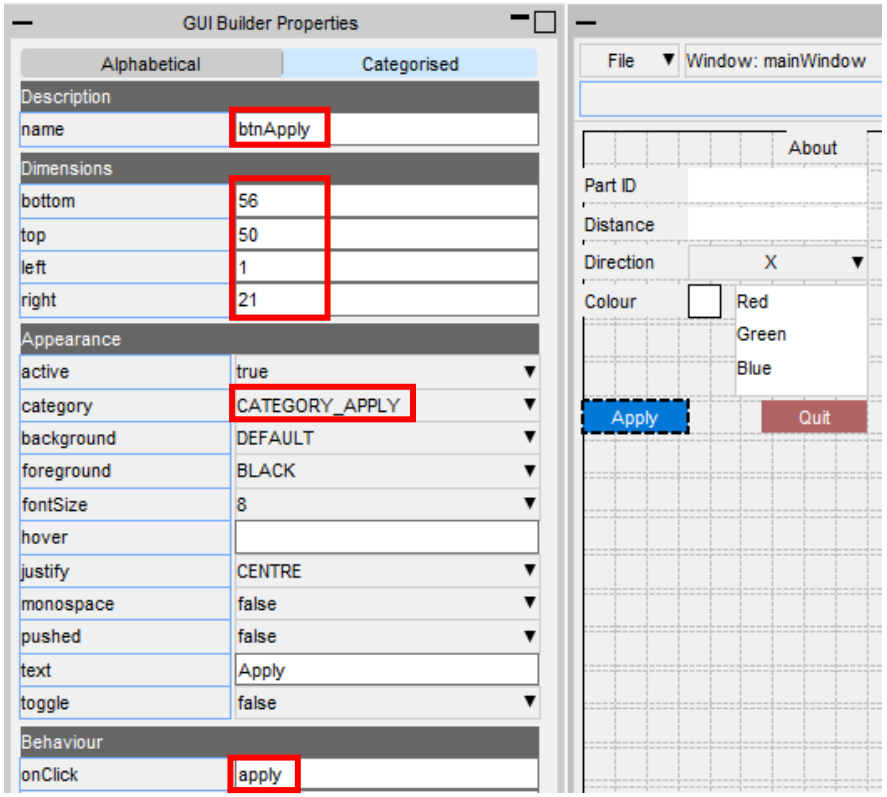


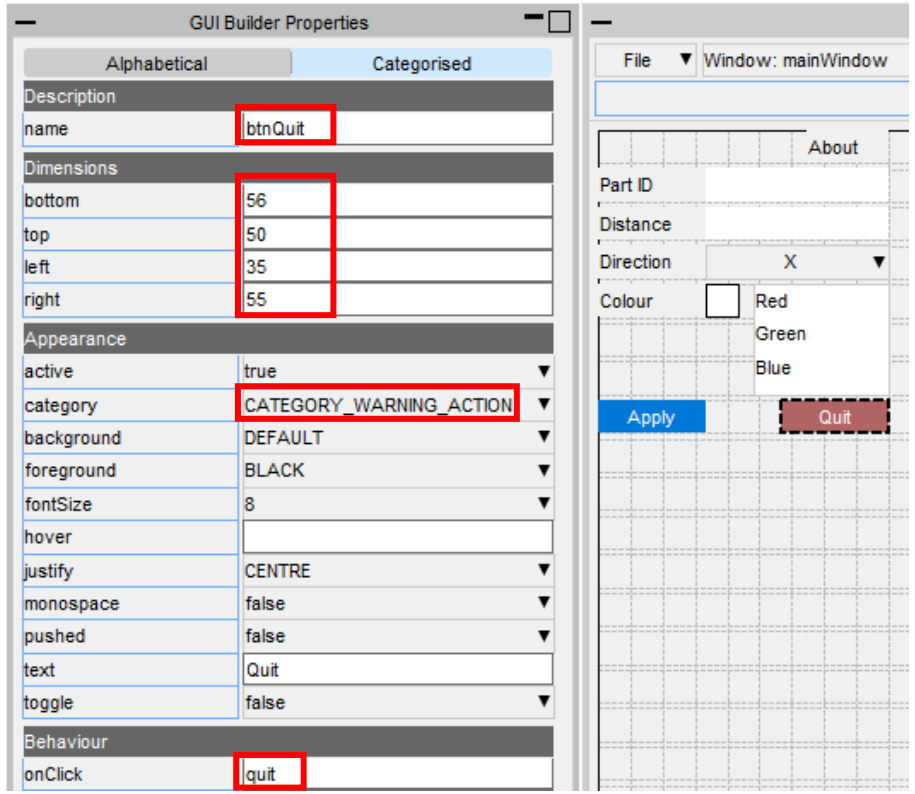


Add Button Widgets

Right-click anywhere in the Design Window and select ‘Insert button’. Do it again to create another Button.

Set the properties of each Button as shown:



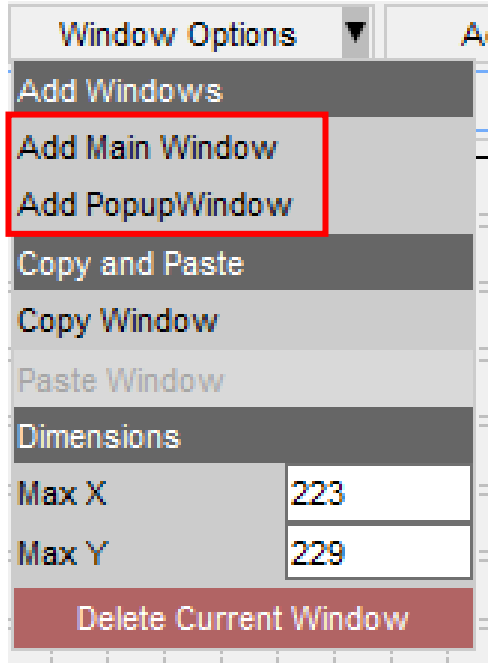


Create additional Windows

Additional Windows

The GUI Builder allows the user to create multiple Windows from the Window Options menu.

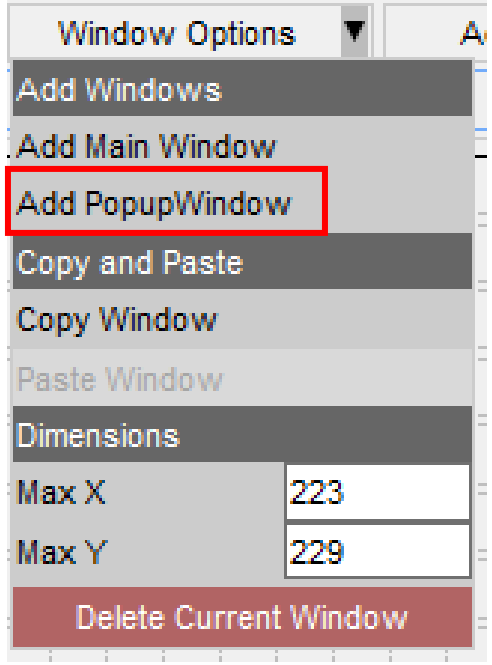
These can be either ‘Main’ Windows (the Window Class in the JS-API) or ‘Popup’ Windows (the PopupWindow Class in the JS-API).

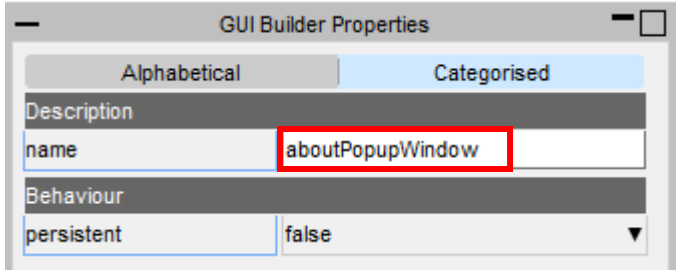


Create a PopupWindow

For this example we're going to create a PopupWindow and then attach it to the 'About' Label.

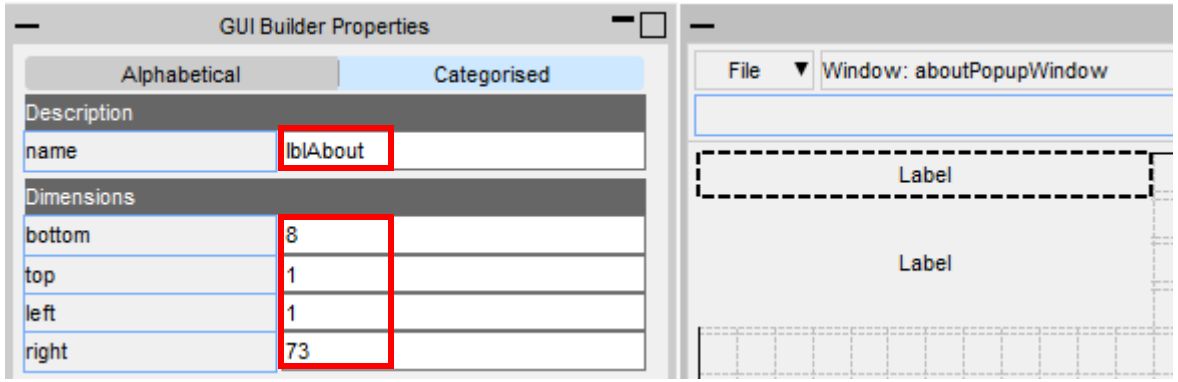
First, click on 'Add PopupWindow' and then set the name to 'aboutPopupWindow'.

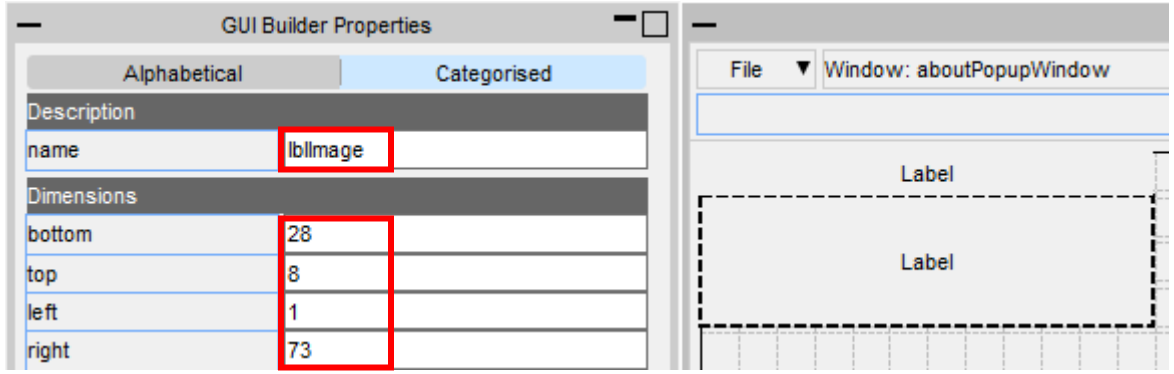




Create a PopupWindow

The popup is going to contain a Label with some text and a Label with an image, so add two Label Widgets to the Window and then set the properties as shown.

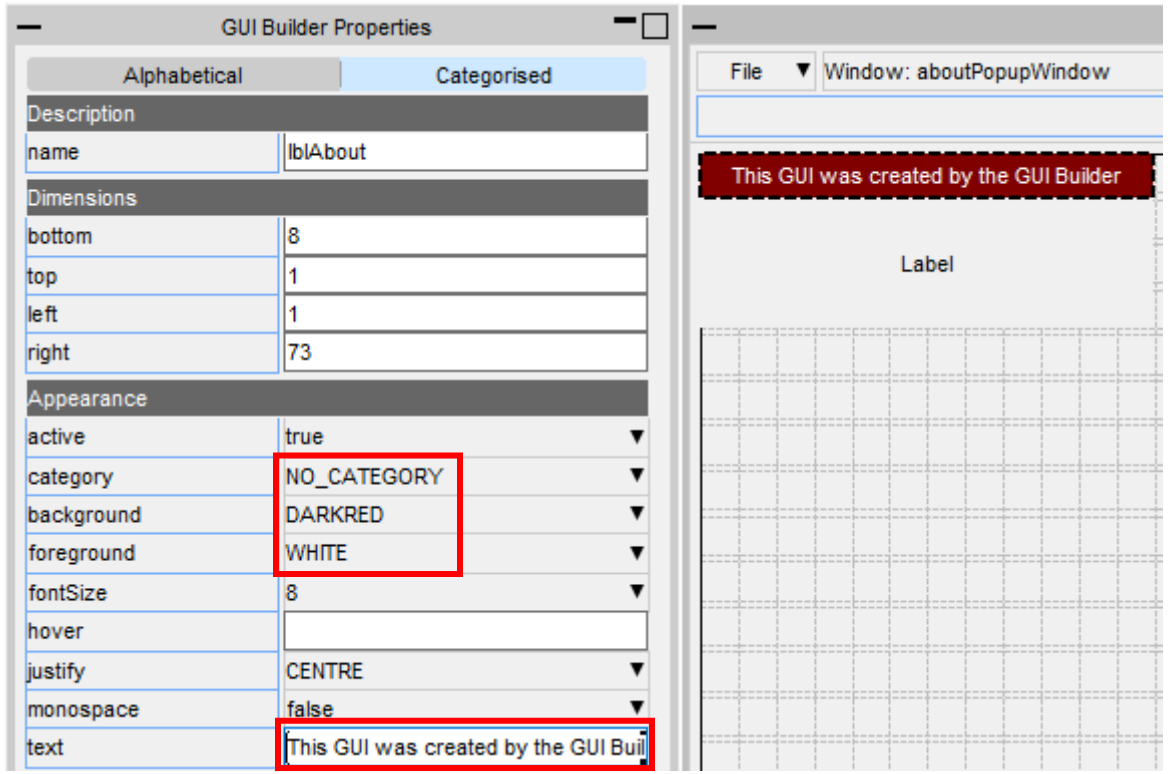




Create a PopupWindow

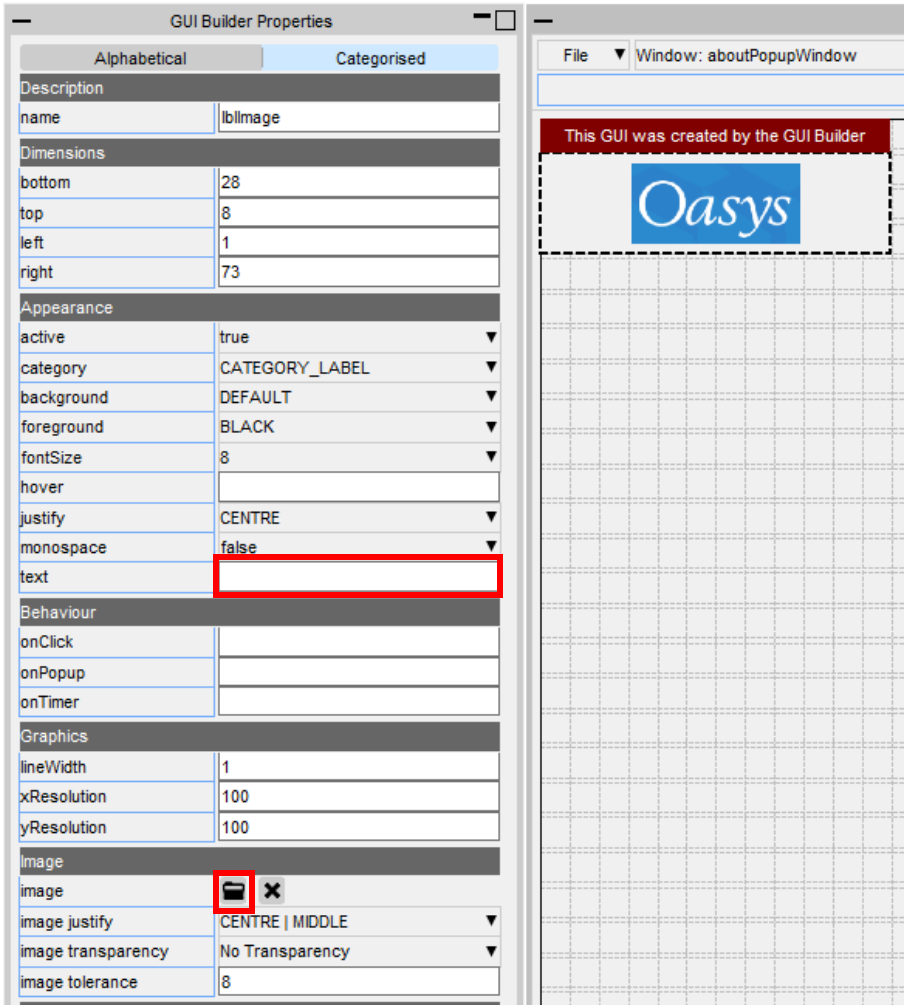
Set the category property of the top Label to NO_CATEGORY.

This means that we can colour the Label using the background and foreground properties. Select some colours and set the text to whatever you want.



Create a PopupWindow

For the image Label, delete the text for the text property and then use the file selector to select an image saved on your machine.

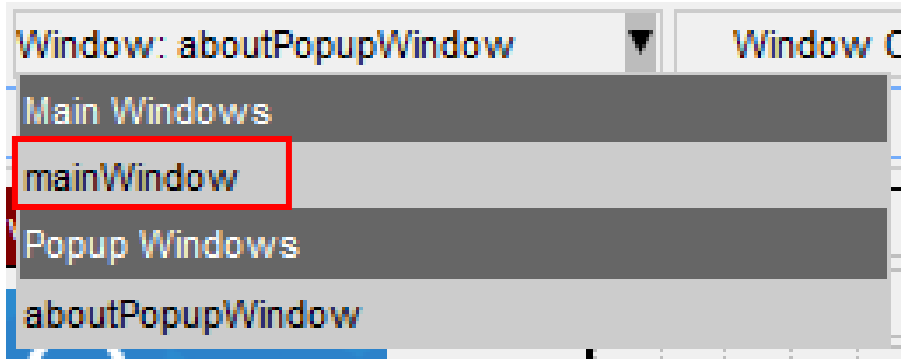


Create a PopupWindow

The PopupWindow is now complete, but we still need to attach it to the ‘About’ Label in the main Window.

The Windows menu at the top of the GUI Builder lists all the different Windows in the GUI and allows you to switch between them.

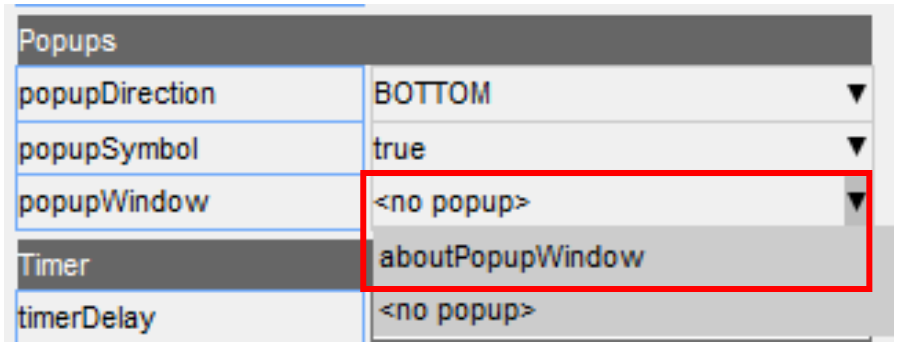
To switch back to the main Window select ‘mainWindow’.

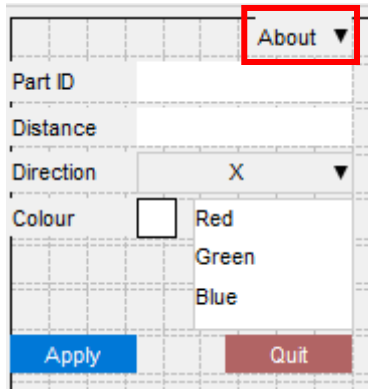


Create a PopupWindow

To attach the PopupWindow to the About Label, left-click on the Label in the Design Window and then select the ‘aboutPopupWindow’ from the popupWindow property dropdown.

The Label should update in the Design Window with an arrow to indicate it’s attached to a PopupWindow





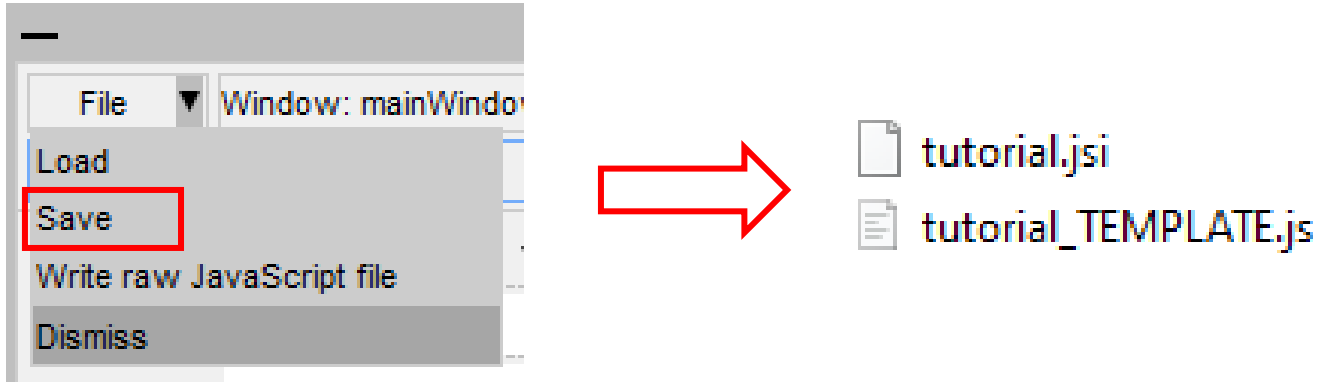
Saving the GUI

Save the GUI

The GUI has now been created and needs to be saved to a file so it can be used in a JavaScript.

Click on File->Save and save it as *tutorial.jsi* (the *.jsi file extension indicates that it is a JavaScript file to be included by another file).

This will output two files, the *tutorial.jsi* file containing the GUI definition, and a *tutorial_TEMPLATE.js* file which reads the *.jsi file via the *Use()* function.



Save the GUI

tutorial_Template.js

```
/*
 * This JavaScript file was automatically created by the Oasys Ltd GUI builder.
 *
 * It demonstrates how to include the file generated by the GUI builder
 * and display the first window.
 *
 * Blank callback functions referenced by windows or widgets, e.g. onClick
 * are also written, ready to be filled in.
 *
 * It is recommended to rename this file so that it doesn't get overwritten
 * by the GUI builder the next time the GUI is saved.
 */

/*
 * Include the file to build the GUI. This sets a global object
 * containing information about any windows and widgets defined.
 *
 * By default the name of the object is 'gui', but it can be changed
 * from the General Options dropdown in the GUI Builder.
 */
Use("tutorial.jsi");

/*
 * The windows have been stored as properties on the global object 'gui'.
 *
 * If there is a window named 'Window1' it can be referenced by:
 *
 *     var wdw = gui.Window1;
 *
 * Widgets are stored as properties on the window object.
 *
 * If there is a widget named 'my_widget' on window 'Window1' it can be referenced by:
 *
 *     var widget = gui.Window1.my_widget;
 *
 * WidgetItems are stored as properties on the widget object.
 *
 * If there is widget item named 'my_widgetitem' on widget 'my_widget' on window 'Window1'
 * it can be referenced by:
 *
 *     var widgetitem = gui.Window1.my_widget.my_widgetitem;
 */

/* Show the first window "mainWindow" */
if (gui) gui.mainWindow.Show();

/* Callback functions */
function updateListbox()
{

}

function apply()
{

}

function quit()
{

}
```

This reads the GUI definition from the *tutorial.jsi* file and stores it in a global object named *gui* by default (this can be changed in the General Options menu in the GUI Builder).

Each Window is stored as a property on the *gui* object. The name of the property is whatever was defined in the properties window in the GUI Builder, i.e. we named our window ‘mainWindow’.

Each callback function is automatically created, however it is up to the user to fill them in so that they actually do something.

Adding code to callbacks

quit callback

PRIMER

We'll start by filling in the *quit()* callback function, which needs to close the window when it's clicked.

Open *tutorial_TEMPLATE.js* in a text editor and update *quit()* with this line to close the mainWindow:

```
function quit()
{
    gui.mainWindow.Hide();
}
```

You should now be able to run the *tutorial_TEMPLATE.js* file from the JavaScript menu. The Window should get displayed and if you press the Quit button it should close.

updateListbox callback

Next we'll fill in the *updateListbox()* callback function. This is called when the Checkbox is clicked on and should make the Listbox active when it's on and inactive when it's off.

Update the function with the following line:

```
function updateListbox()
{
    gui.mainWindow.lbxColour.active = gui.mainWindow.chkColour.pushed;
}
```

This shows how to access Widgets from the *gui* object. Just as Windows are properties of the *gui* object, Widgets are properties of the Windows, with the names defined in the GUI Builder used as the property names, e.g. the Listbox Widget is on the mainWindow and was named 'lbxColour' so is accessed by *gui.mainWindow.lbxColour*.

apply callback

PRIMER

The *apply()* callback function is a bit more complicated than the other two. It needs to move the selected part the specified distance, in the specified direction and set the colour if selected by the user.

The next few slides shows the code that needs to be added and describe what it's doing. You should be able to copy and paste this into your own file.

apply callback

```
function apply()

{

    /* Move the selected part and colour it if selected */

    var m = Model.First();  /* Assume there is at least one model loaded */

/* Get the details needed to move the part */

/* Get the Window object, to save having to type gui.mainWindow for each
Widget */

    var win = gui.mainWindow;

    var partId = parseInt(win.txtPartId.text);

    var distance = parseFloat(win.txtDistance.text);

    var direction = win.cbxDirection.text;

    /* Flag the part, propagating it to it's nodes */

    var flag = AllocateFlag();

    var part = Part.GetFromID(m, partId);

    part.SetFlag(flag);

    m.PropagateFlag(flag);
```

Get the model (assumes there is a model loaded)

Get the values input by the user from the Widgets

Flag the selected Part and propagate the flag to Nodes

Continued on next slide…

apply callback

```
/* Move all flagged nodes */

    var n = Node.First(m);

    while(n)

    {

        if (n.Flagged(flag))

        {

            switch (direction)

            {

                case "X": n.x += distance;  break;

                case "Y": n.y += distance;  break;

                case "Z": n.z += distance;  break;

            }

        }

        n = n.Next();

    }
```

Move all the flagged nodes the specified distance in the specified direction

Continued on next slide…

apply callback

```
/* Set the colour if selected by the user */

    if (win.chkColour.pushed)

    {

        var widgetItems = win.lbxColour.WidgetItems();

        for (var i=0; i<widgetItems.length; i++)

        {

            var wi = widgetItems[i];

            if (widgetItems[i].selected)

            {

                var colour = wi.text;

                switch (colour)

                {

                    case "Red":   part.colour = Colour.RED;    break;

                    case "Green": part.colour = Colour.GREEN;  break;

                    case "Blue":  part.colour = Colour.BLUE;   break;

                }

            }

        }

    }
```

Colour the Part, if the Checkbox was selected.

This then loops over the WidgetItems to work out which colour to use.

Continued on next slide…

apply callback

```
    /* Redraw and return the flag */
    m.UpdateGraphics();
    ReturnFlag(flag);
}
```

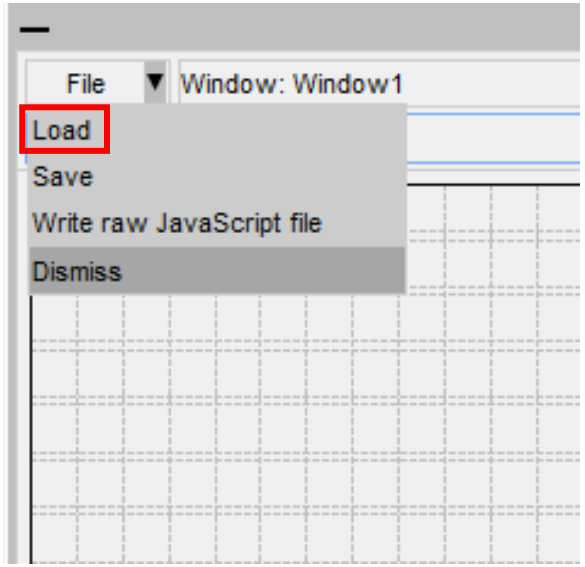
Redraw and tidy up.

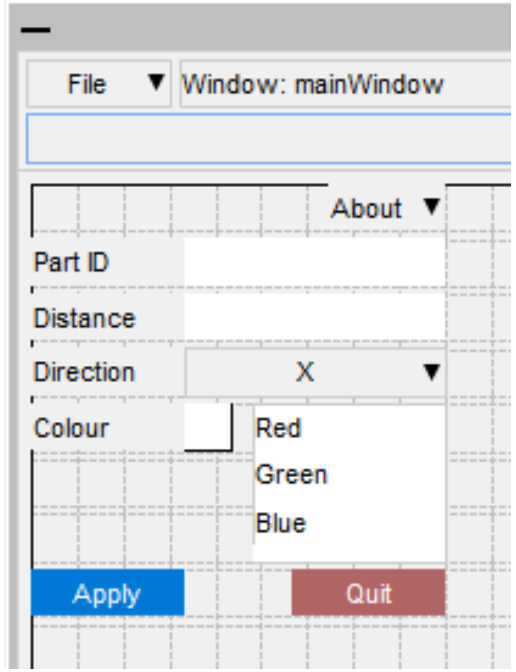
If you run the *tutorial_TEMPLATE.js* script you should now be able to select a part, specify a distance to move it, the direction to move it and optionally colour it when you press the Apply button.

Other

Reloading a GUI

The *.jsi file that contains the GUI definition can be read by the GUI Builder to reload a GUI at any time.





Write GUI as raw JavaScript

PRIMER

The *.jsi file contains a special JSON string that contains the GUI definition. This is read when the file is included from another file and generates the Windows, Widgets and WidgetItems. However this can only be used in from v18 of the Oasys software onwards.

If you want to use the GUI in earlier versions of the software (perhaps for legacy reasons) then it can be saved as a raw JavaScript file instead. However, be aware that this file *cannot* be read back into the GUI Builder.

Contact us

Oasys
LS-DYNA Environment

Global / UK
T: +44 121 213 3399
E: dyna.support@arup.com

India
T: +91 40 69019723 / 98
E: india.support@arup.com

China
T: +86 21 3118 8875
E: china.support@arup.com

USA
T: +1 415 940 0959
E: us.support@arup.com

Subscribe to
our newsletter:

Follow us on:

@Oasys LS-DYNA Environment

@Oasys LS-DYNA Environment

@Oasys

@Oasys

www.oasys-software.com/dyna/