Balise manual

François Guillet

Table of Contents

Installation

First, download and install Buoy if it is not already installed on your system. Next, untar the Balise zipped tar archive (.tgz) where you want to install it. Balise saves files between different sessions in a folder chosen by the user. A sample folder is provided in the folder named `BaliseFiles`, which can be found in the Balise root directory. You may either keep this `BaliseFiles` folder where it is and use it, use a brand new folder anywhere else or copy the `BaliseFiles` folder anywhere you want. However, note that the default `BaliseFiles` folder clipboard files contain useful widgets like OK and Cancel buttons.

A standalone instance of Balise is launched using the following command, current path in Balise root directory:

java -cp /path_to/Buoy.jar:Balise.jar balise.Balise (Linux)

java -cp C:\path_to\Buoy.jar;Balise.jar balise.Balise (Windows)

Alternatively, you can attach a balise editor window to any WindowWidget instance using the following call in your source code :

```
window = new BFrame( "a window" );

setContent( new BorderContainer() );

[...]

WindowEditor we = new WindowEditor( window );
```

Please note that the edited window *must have at least one widget container set as content*.

The first time Balise is launched, it asks for the location of the Balise files folder. Choose any location that suits you, as indicated above. You will be able to change that location any time afterwards. If Balise crashes right after setting the files location, relaunch it (and keep me informed !). If you choose afterwards a new location to store balise files, you will loose memory of Balise preferances, windows locations and sizes as well as permanent widgets in the clipboard.

Getting Started

In order to illustrate how an interface is built using Balise, we will build a classical "Grid properties" dialog, such as the one that can be found that sets the Explicit Container Grid properties. This dialog is shown in figure Figure 1. This diagram also shows all the widget containers and the widgets this dialog consists of. The root level of the dialog content is a column container. Here are the different items that are contained within this column container, from top to bottom:

- An outline that encloses a grid, which itself contains in each of its two cells a set of one label and one spinner to set the horizontal and vertical grid spacings.
- A grid also made of two cells, one cell for the Show Grid check box and one cell for the Snap to Grid check box.
- The usual OK and Cancel buttons.

Launch Balise using the command given in the installation section. You should see two windows, the widget palette and the clipboard.

The widget palette mainly consists in a tabbed pane that show two tabs, a widget tab and a widget container tabs. The content of these two tabs is shown in figure Figure 2 and Figure 3. Select one of these buttons to determine the type of the next widget that you will add in your interface. The currently selected widget is shown in the bottom right corner of the widget palette. There is only one instance of this window, whatever the number of window widgets currently being edited. For this reason, the widget palette can also be seen as Balise main window. New windows are loaded or created using the widget palette Window menu. Using this menu, you can choose to create a new window (or dialog) or load an XML definition of a `WindowWidget`. If the widget described in the XML file is not a window, Balise will create a window and set the widget as its content (or menu bar if the loaded widget happens to be a `BMenuBar`).

The clipboard holds the items you copied or drag'n'dropped into it (which amounts to the same). There is much to say about the clipboard, which also acts as a provider for standard or favorite widgets: be sure to read the section about the clipboard.

The first thing to do is to create a new dialog using the command Create new BDialog from the Window menu of the widget palette. An empty dialog appears, along with an editor window. Both should look as shown in figure Figure 4. The widget and widget container hierarchical structure is mirrored in the widget tree on the left side of the editor window. Selecting a widget in this tree also selects this widget for editing purposes. The browser-like navigation button at the top of the window allows to recall recently edited widgets (left and right arrows), edit the parent widget container of the currently selected widget (up) or go straight to the content widget (home). Most of the edition process takes place at the left (or more appropriately center) part of the editor window which consists of a tabbed pane.

By default, a newly created window displays a border container as content (this is a fairly common situation). Balise always assumes that a window is not empty : you cannot delete the content widget. You may switch it to another widget container type, paste a widget in place of the content widget, embed the content widget in another widget container, etc., but *you cannot delete the content widget*. Given all the possibilities you have to modify the content widget, this was not deemed necessary. The contents of the default border container are shown in the Contents tab. A set of buttons or thumbnails depicts the current state of the border container. Whenever a position (CENTER, NORTH, etc.) is occupied by a widget, the button turns green. More about these buttons later : for the time being, the content widget should be a column container, not a border container. Select Column in the combo box at the bottom right of the content tab, and click on the switch button. Watch the widget tree: it should now state ColumnContainer1 as being the content widget. The Contents tab now shows only one button.

It's high time we start to fill the content widget. The top widget is a `BOutline`, but we will skip it for educational purposes (let's pretend we forgot about that one). What we need is then a grid container. Select the Widget Containers tab in the widget palette and then select the GridContainer button. Next, click on the red button in the Contents tab of the editor window. A grid container widget should now be attached to the column container in the widget tree, and the editor window should look as the window shown in figure Figure 5.

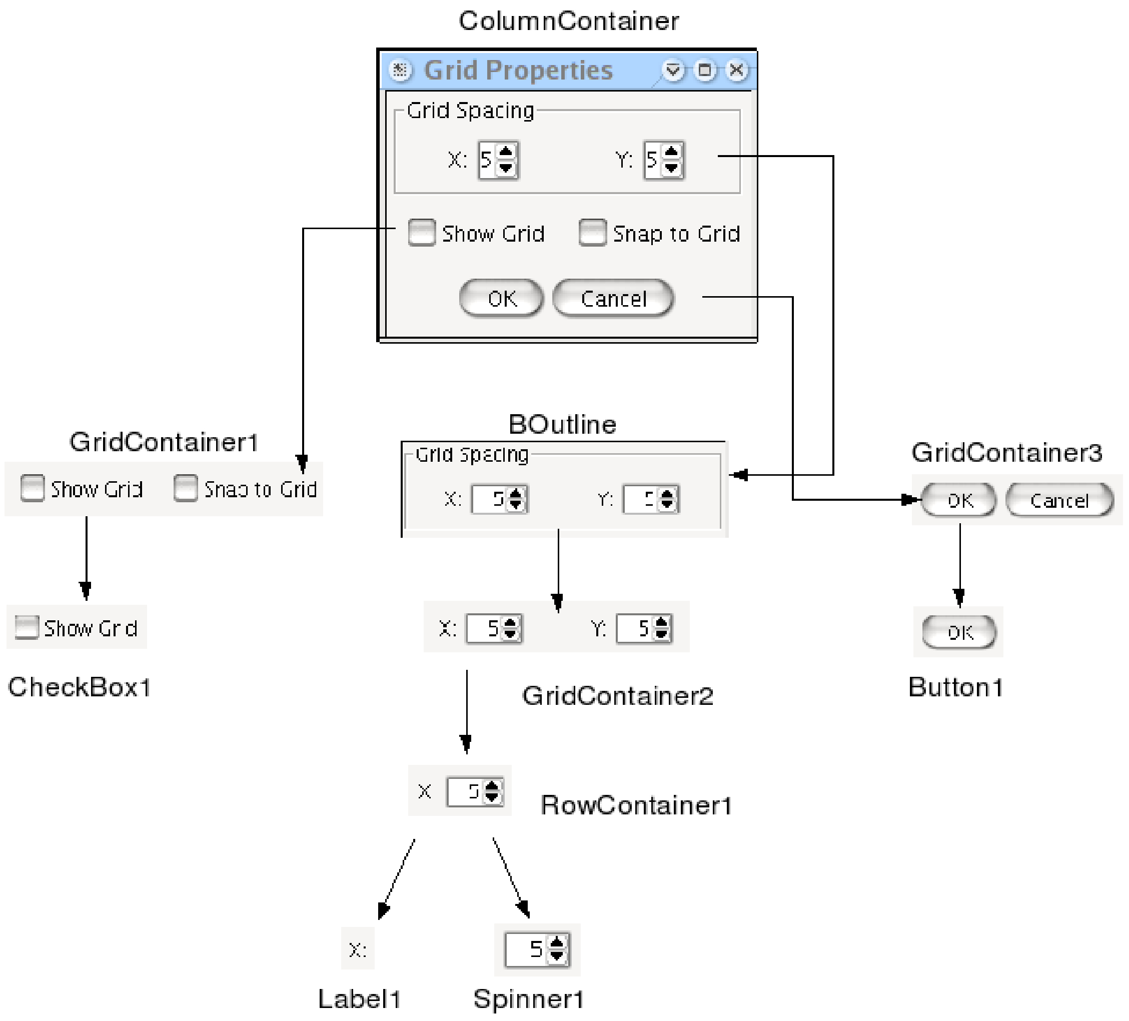


Figure 1. Diagram of the Grid properties dialog structure

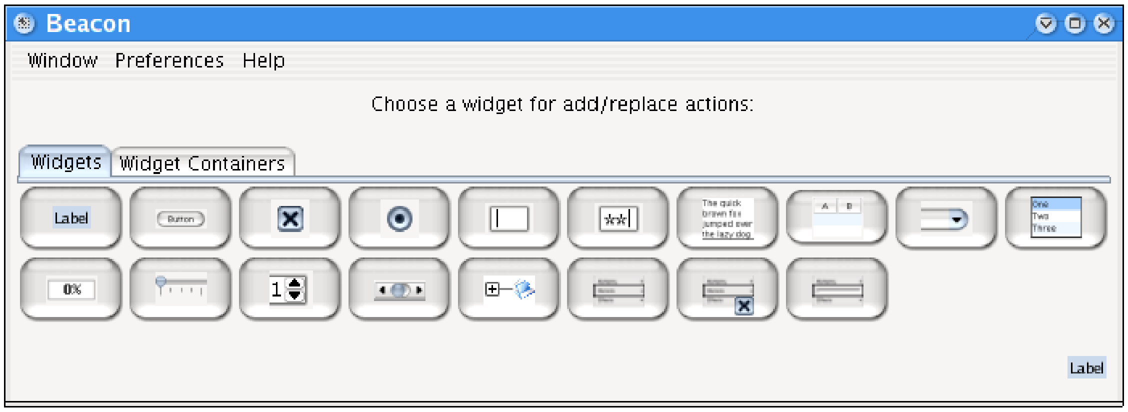


Figure 2. The widget palette, Widget tab

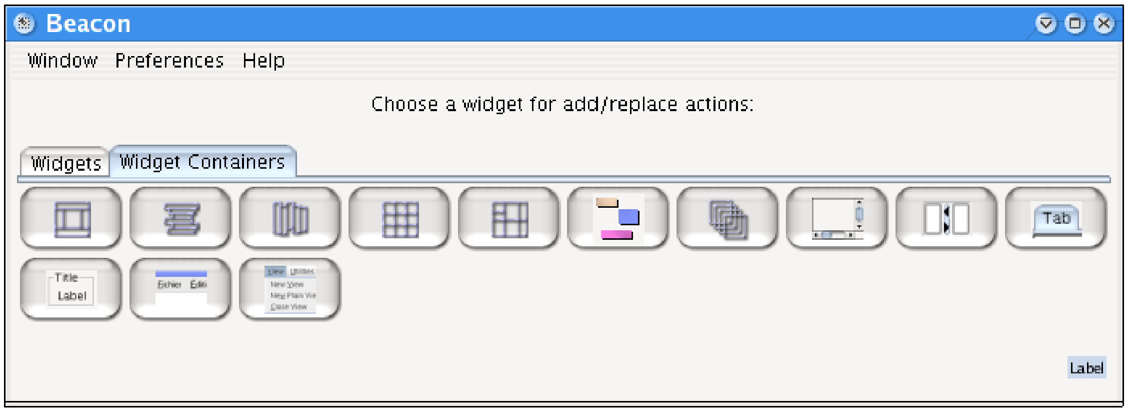


Figure 3. The widget palette, WidgetContainer tab

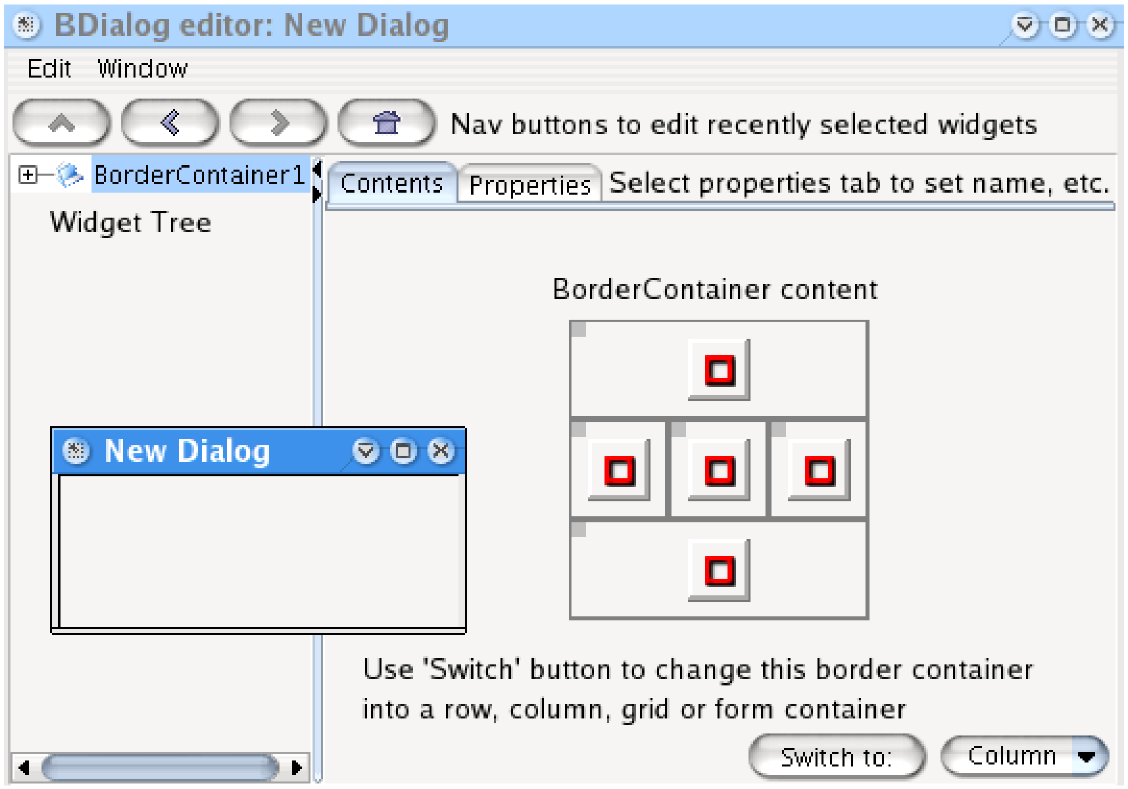

BDialog editor: New Dialog
Edit
Window
Nav buttons to edit recently selected widgets
BorderContainer1
Contents
Properties
Select properties tab to set name, etc.
Widget Tree
BorderContainer content
New Dialog
Use 'Switch' button to change this border container
into a row, column, grid or form container
Switch to:
Column

Figure 4. The editor window

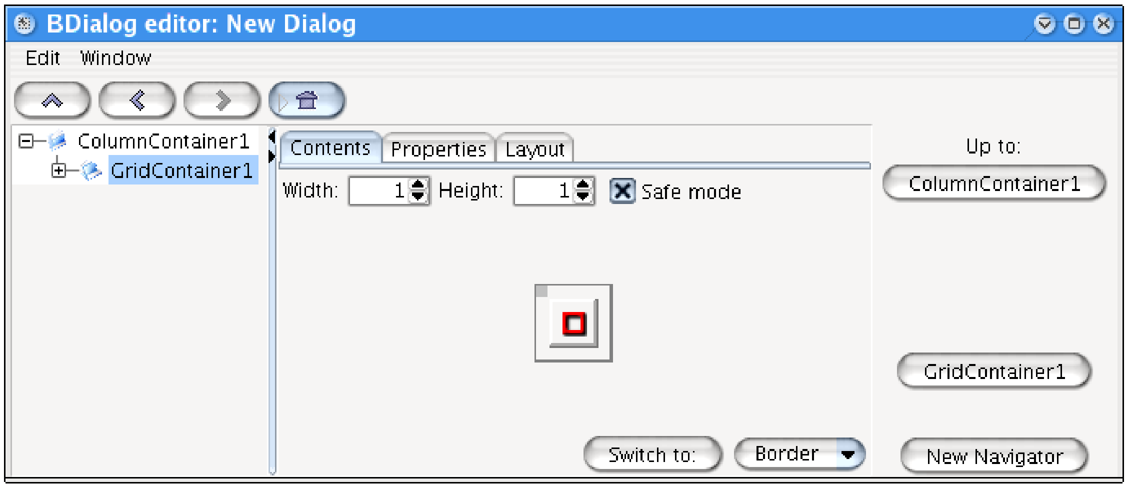


Figure 5. The editor window : grid container editing

You should notice a newcomer in the right part of the editor window: a set of navigation buttons has appeared. These buttons allow to quickly select (and edit) any other child of the column container widget (the grid container parent widget). The 'up' button here labeled ColumnContainer1 also allows to edit the column container itself. The button labeled New Navigator allows to get a copy of this set of buttons in a standalone windows. If you have a major or often used widget container in your GUI component, you can ask for a detached navigator and then easily recall any child of this widget container (or the widget container itself). Last but not least, another way of selecting a widget is simply to click on it in the edited window. A black frame will outline the widget to show it has been selected. If you click several times in a row (within 0.7s to be precise -no need to hurry !-), say n times, the nth-1 parent widget container of the clicked widget will be selected instead of the widget itself. This way, parent widgets can be selected even if they do not provide a 'clickable' space because their children take up all the available space.

The grid container editor Contents tab shows more items than the border container editor did. The width and the height of the grid container can be set using the two relevant spinners. A check box allows to resize the grid in a 'safe' mode, where any downsizing that will cause the loss of at least one widget will trigger user confirmation through a dialog. Another feature concerning grid shrinking is that 'lost' widget will be retained as much as possible in the new layout. For example, suppose a 3x3 grid has a child at (3,3), and it's shrunk to (2x2). If the (2,2) position is unoccupied, the widget at (3,3) will be sent to (2,2) prior to shrinking. The user has to manually delete this widget if he does not need it anymore. If there is no way the (3,3) widget can be kept it is simply discarded (safe mode disabled) or the user is asked to confirm deletion (safe mode enabled).

The grid container navigator (the set of buttons on the right) identifies the grid children using numbers instead of widget names. This is because using names for a grid layout can use more space than reasonable for a navigator. Numbers make shorter buttons, although they make child widget anonymous.

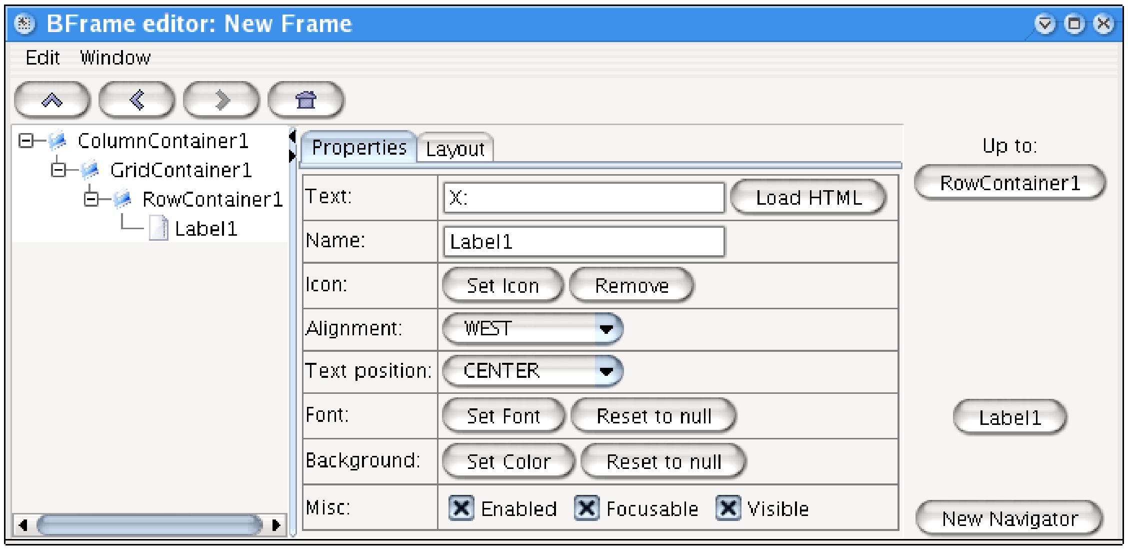
Back to the tutorial : we need two horizontal cells (one row, two columns), so set the width to two. Each of the cell will be occupied by a row container, so choose the RowContainer button in the widget palette and click on the first red button of the grid container editor. The row container editor looks a lot like the column container editor, so there is nothing more to point out.

We now have to add a label and a spinner to the row container. Get back to the widget palette and select the BLabel button. Click on the red button of the row container editor. The editor window should now look like figure Figure 6. This time, no Contents tab is seen in the tabbed pane: a label is a `Widget,` not a `WidgetContainer`, and has no content (child widgets). The Properties tab allow to set the properties of the label, more or less from the most useful to the most seldom used ones. The most important property, of course, is the text of the label. Enter X: in the text field. You don't need to modify the name of the widget because you won't need to access the label (as a variable) at run time. Otherwise, it is a good idea to give a widget a more meaningful name than just 'Label1'.

Get to the parent row container, either using the up arrow in the browser-like set of nav buttons at the top of the editor window, the RowContainer1 button of the navigator at the right or pressing the up arrow key. You should now see that the Contents tab shows two buttons instead of one : one green and one red. The green button identifies the label that has been added. The red button identifies an empty position where widget addition takes place. The row container editor, the column container editor and some other like explicit container editor show a "trailing" empty button where widgets may be added.

Select the `BSpinner` button in the widget palette and add a spinner through clicking the empty button. The spinner editor window a bit complex because a spinner may use integer values, double values, a list of predefined values, or dates. The default integer spinner is exactly what we need : we just have to set a minimum value of 1 (a zero grid spacing means trouble!) and a maximum value of, say, 1,000. The current value can reasonably be set to 5 (the program will probably set this value to the real value after loading the GUI, anyway). Tip: the spinner might calculate its width from its initial value. A value of 5 means that the spinner might be quite narrow and display only one figure (5). A trick is to use a 2 or 3 figure initial value (like 100) to force the spinner show 2 or 3 figures and have the program set the correct value after loading and packing the interface. This time, it's best to give the spinner a meaningful name like xSpinner or XSpinner.

Let's get back to the GridContainer. Instead of going through the process of creating another row container with its label and spinner, we will simply copy the row container and paste it in the second cell. Select the first green button not by clicking on the button but on the frame around the button. The frame should now be outlined in light purple as shown in figure Figure 7. Copy the button using Copy from the Edit menu (or its **Ctrl-C** shortcut), or drag the button using the blue square and drop it onto the clipboard, using the **Ctrl** key to avoid deleting the widget as it is copied. The edit menu shortcuts also work in the edited window. A new item should appear in the clipboard, as shown in figure Figure 8 (actual widget location may vary). The green frame around the copied widget means that it is selected and that it is a non persitent item. Non-persistent item are discarded when Balise quits. Persistent items are displayed with an italic face and show a red frame when selected. They are kept between different Balise sessions. If there is a particular widget you often use (like the OK and Cancel buttons provided in the buttons tab), copy it to the clipboard and make it persistent. New tabs can be added to the clipboard for widget identification.



Text:	X:
Name:	Label1
Alignment:	WEST
Text position:	CENTER
Misc:	[✓ Enabled] [✓ Focusable] [✓ Visible]

Figure 6. Editor window : label editing

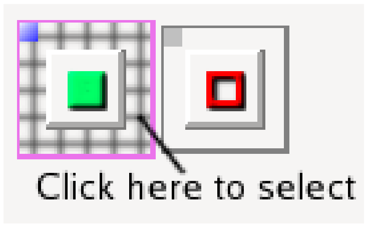


Figure 7. Editor window : Selecting a content button

Now select the second red button of the grid container and paste the copied widget, or alternatively drag it from the clipboard onto the red button. If you use plain drag and drop, the widget will be deleted from the clipboard. As before, hold down the **Ctrl** key if you want to prevent the widget from being deleted (this is not useful here since it is unlikely that the row container will be reused). Edit the newly created label and spinner and set the label text to Y: and the spinner name to YSpinner. The edited window should now look as figure Figure 9.

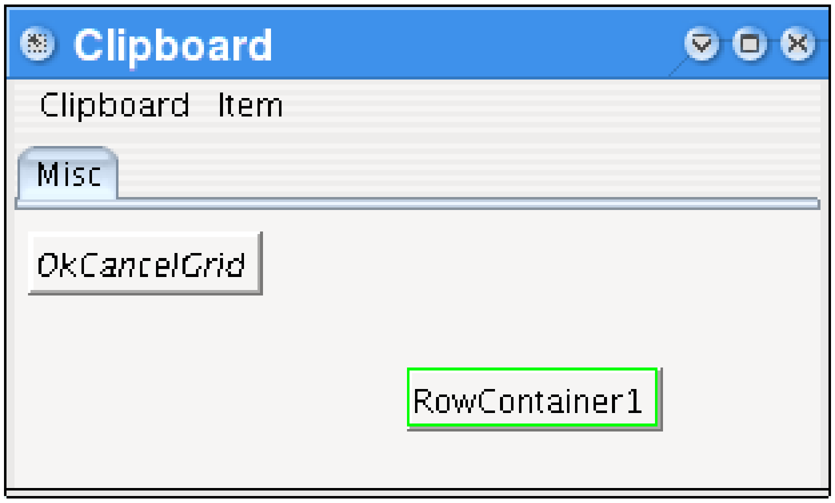


Figure 8. The clipboard after selecting the row container

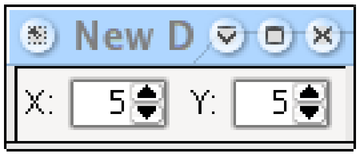


Figure 9. The edited window after completing grid edition

Get back to the main column container, select the `GridContainer` widget container in the widget palette and add a grid container at the bottom of the column container (bottom red button). Make this new grid 2 cells wide and add a `BCheckBox` in each of the cell. If you want to add several widgets in a row whithout automatically selecting them as they are created, hold down the **Ctrl** key while selecting a red button. Set the text of the first checkbox to Show Grid and the text of the second checkbox to Snap to Grid. Name the two checkboxes as you wish, e.g. ShowGridCB and SnapCB. The resulting window is shown in figure Figure 10.

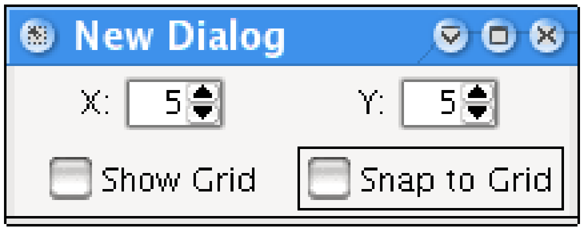


Figure 10. The edited window after adding the two checkboxes

The next step consists in adding the OK and Cancel buttons. For illustration purposes, we will suppose that you used the sample folder supplied with Balise as Balise Files folder, in which case the clipboard definition contains a persistent GridContainer with these two buttons. Drag this grid container in the empty red button of the column container. Note that persitent clipboard items are not deleted after being dragged and dropped, whereas non-persitent items are deleted if the **Ctrl** key is not held down during the process. The names and texts as copied from the clipboard are just what we want, so there is no need to edit them. The edited window should look as shown in figure Figure 11.

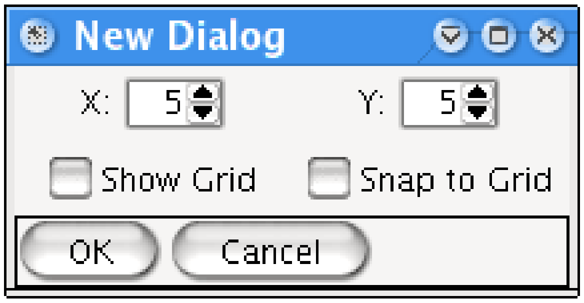
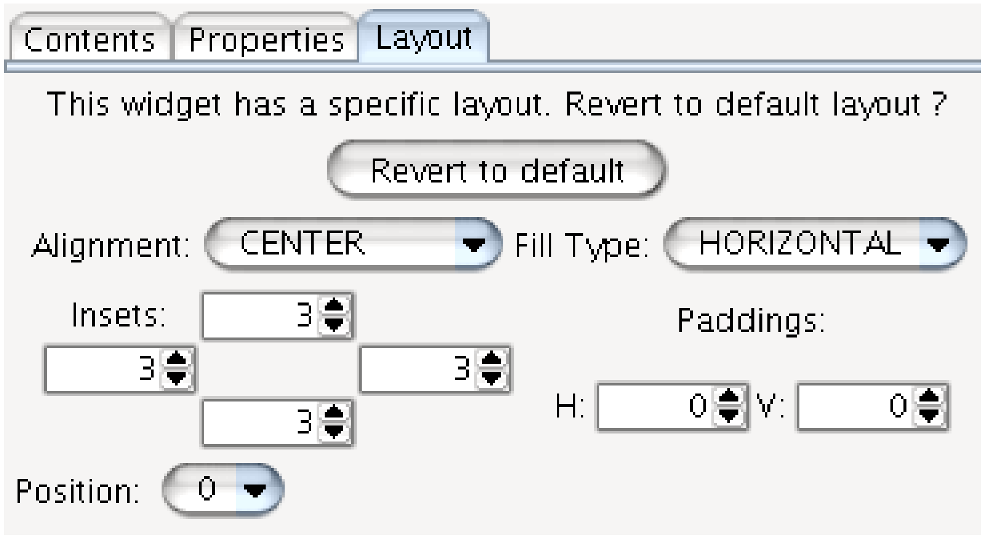


Figure 11. The edited window after adding the OK and Cancel buttons

The basics of the grid properties dialog are there, but there is obviously a need for additional layout... First, the top grid should be outlined. Select the top grid, select the outline container in the widget palette and select the Embed in Container menu item in the Edit menu of the editor window. The grid is now embedded in an outline container. Select the Properties tab of the outline container editor, which is fairly complex due to the many possibilities offered by Swing in terms of borders. For the time being, select a Title border and enter Grid Spacing as title text. It is also a good idea to specify a different layout for the outline widget than the column container default layout. Go to the Layout tab. Since the outline container layout corresponds to its parent default layout, the layout parameters widgets are disabled. Select the Create Layout button : a layout is created and the parameters widgets are now enabled. Keep a CENTER layout, select an HORIZONTAL fill and enter (3, 3, 3, 3) as layout insets, as shown in figure Figure 12. Similarly, select the grid container that contains the two checkboxes and create a new layout. Specify a CENTER layout with a NONE fill type. Finally, select the buttons row container and specify a CENTER layout, NONE fill type and (10, 0, 5, 0) insets (10 and 5 corresponds to top and bottom values).



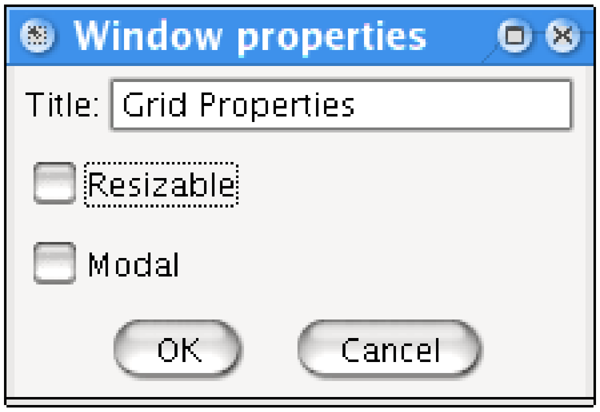
This widget has a specific layout. Revert to default layout ?

Alignment: CENTER
Fill Type: HORIZONTAL
Insets: 3 3 3 3
H: 0 V: 0
Position: 0

Figure 12. The outline layout

All there is now left to do is to specify a title for the dialog. Select the Window properties menu item from the Window menu of the editor window. Enter "Grid Properties" as window title and deselect the Resizable check box if you wish (see figure Figure 13). Please note that the modal setting is ignored at this time. You can test how the window behaves by selecting the Set Run Time Mode menu item in the

Edit menu of the editor window. In this mode, widget selection is ignored. You can also have limited interaction with the edited window widgets in Edit mode if you keep the **Ctrl** key pressed while selecting a widget with the mouse. This feature is not equivalent to interacting with the edited window in run time mode, but it allows for example selection of a tabbed pane tab.



Title: Grid Properties

Resizable

Modal

Figure 13. The window properties window

Though it's not mandatory, it's best to create interfaces within the context of a project. Projects are explained in detail later on. For now, let's just say a project is a directory structure that holds buoy interfaces files, images used in these interfaces and balise specific files. To associate a project to an interface, select the New Project command from the Project menu. This command first asks to save a project file and then opens the project settings window as shown below. It is recommended that the files are organized the way the description of which follows (though it is perfectly possible to use another design). A root folder (say foo) contains three subfolders :

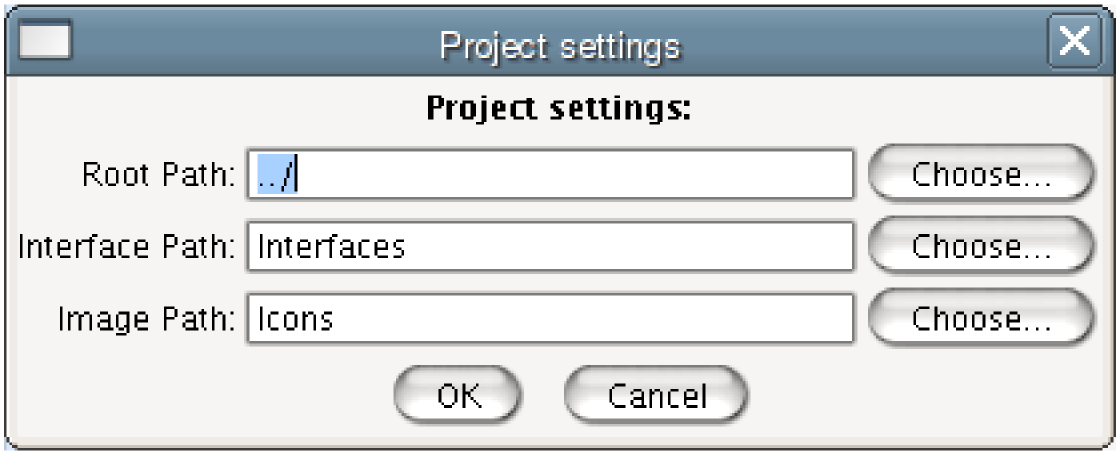
- `BaliseFiles`
- `Images` (or `Icons`)
- `Interfaces`

The first folder will contain Balise specific file such as the project and interfaces files. It is recommended that project files use `.bpr` extension whereas interface files should use `.bui` extension.

The second folder will hold images used by widgets. These resources are external and are not stored in the interface definition files.

The third folder will contain buoy xml files which are updated each time a balise files to which a project is associated is exported. If you don't use a project when editing an interface file, then you must manually export the buoy file.

These folders are set in the Project Settings edit window :



Project settings:

Root Path: ../
Interface Path: Interfaces
Image Path: Icons

Figure 14. The Project Settings Edit Window

The root path is the location of the `foo` folder relative to the `BaliseFiles` folder where the project is saved. The other two are straightforward.

The final stage is actually saving the balise file and use the buoy interface in your program. There are two ways you can do that :

- save the content widget, declare a class that extends BDialog, load the content widget definition file in the constructor and set it as the dialog content.
- save the whole dialog and load it anywhere in your source code.

If you choose the first option, check Save = Export Content Container in the Window menu. If you choose the second option simply leave this option unchecked. It is also possible to prototype a single selected widget or widget container using the Prototype Widget Code command. In this tutorial, we will assume you chose to save the whole window. Balise helps you write the code to load the window from its definition file : select the Prototype Window Code menu item from the Window menu of the editor window. The following prototype code is displayed. You can copy any part of it and incorporate it in your source code. Balise does not manage or modify any of your source code file (and probably never will, I don't like much prototyping). This prototyped text just saves typing when designing or modifying a GUI. It is possible to specify in each widget properties window if a widget should appear in the prototyped code.

```
InputStream inputStream = null;
try
{
    inputStream = new FileInputStream( new File("Interfaces/test.xml") );
    WidgetDecoder decoder = new WidgetDecoder( inputStream );
    BDialog window = (BDialog) decoder.getRootObject();
    ColumnContainer columnContainer1 = ((ColumnContainer) decoder.getObject("Colum
    BOutline outline1 = ((BOutline) decoder.getObject("Outline1"));
    GridContainer gridContainer1 = ((GridContainer) decoder.getObject("GridContain
    RowContainer rowContainer1 = ((RowContainer) decoder.getObject("RowContainer1"
    BLabel label1 = ((BLabel) decoder.getObject("Label1"));
    BSpinner xSpinner = ((BSpinner) decoder.getObject("XSpinner"));
    RowContainer rowContainer1 = ((RowContainer) decoder.getObject("RowContainer1"
    BLabel label1 = ((BLabel) decoder.getObject("Label1"));
    BSpinner ySpinner = ((BSpinner) decoder.getObject("YSpinner"));
    GridContainer gridContainer1 = ((GridContainer) decoder.getObject("GridContain
    BCheckBox showGridCB = ((BCheckBox) decoder.getObject("ShowGridCB"));
    BCheckBox snapCB = ((BCheckBox) decoder.getObject("SnapCB"));
    RowContainer rowContainer3 = ((RowContainer) decoder.getObject("RowContainer3"
    BButton okButton = ((BButton) decoder.getObject("okButton"));
    BButton cancelButton = ((BButton) decoder.getObject("cancelButton"));
    window.pack();
    window.setVisible( true );
}
catch(IOException ex)
{
    ex.printStackTrace();
}
finally
{
    try
    {
        if (inputStream != null)
            inputStream.close();
    }
    catch(IOException ex)
    {
        ex.printStackTrace();
    }
}
```

Balise Windows

The Widget Palette

At first glance, the widget palette job is to allow choosing between different widgets or widget containers. However, it is also the Balise key window and does more than just widget type selection. A widget palette singleton is shared among instances of editor windows: for this reason all editor windows independant functions have been placed in the widget palette. The widget palette is thus used to access user preferences and Balise about box.

The widget palette content is straighforward : each widget class is associated to a button. When a button is clicked, the associated widget class becomes the current class for all "widget addition" operation. The currently selected widget icon is draw at the bottom right corner of the widget palette.

Window menu items:

New Frame	Creates a new `BFrame` and opens it for edition.
New Dialog	Created a new `BDialog` and opens it for edition.
Open File...	Asks the user to choose a Balise file.
Import Widget...	Asks the user to choose a buoy definition file. If the root widget is not a frame or a dialog, then the widget is embedded in a BFrame.
Quit	Do I really need to tell you? Warning, though : Balise does not warn for modified files before quitting.

Preferences menu items:

Select Balise Files Folder	Allows the user to choose the folder where all Balise persistent files will reside. This includes properties as well as persistent items in the clipboard.
Select Icons Default Folder	Sets a default folder to choose icon from. This folder is not to be confused with projects image or icons folder. It is just the folder that will be selected by default when choosing an image for an icon.

Help menu items:

Only the "about" dialog is available at the moment.

The Editing Window

The editing window is the window where widget properties and layouts are modified.

This window displays four areas (see figure ???).

At the top, a navigation bar allows to navigate (recall) among the previously selected widgets, much like a browser. Left and right arrows will select widgets according to the selection history. The up arrow will select the parent of the currently selected widget, if possible. The home icon will select the content widget.

On the left, a tree allows to select the widget being edited. The tree root is usually the content widget of the widget. A second node appears at root level if the edited window has a menu bar (as is the case in the example shown in the figure).

A tabbed pane at the middle of the window allows to set properties and layout of the selected widget. If this widget is a container, then it is also possible to set its content using the Contents tab. More about this later on.

A Navigator section is displayed at the right of the window. This section appears only when the parent of the selected widget is a container widget. In this case, one often wants to select a neighbor widget, like if one is currently editing the center widget of a border container and then wants to edit the north widget. The navigator displays several buttons which will directly select a given child widget of the parent of the selected widget. The buttons layout reflects as much as possible the parent widget children layout.If you find yourself constantly referring to a particular parent widget, you may want to detach a navigator to have it constantly at hand. To do so, click on the New Navigator button. Finally, the button just under the Up to: label will select the parent widget.

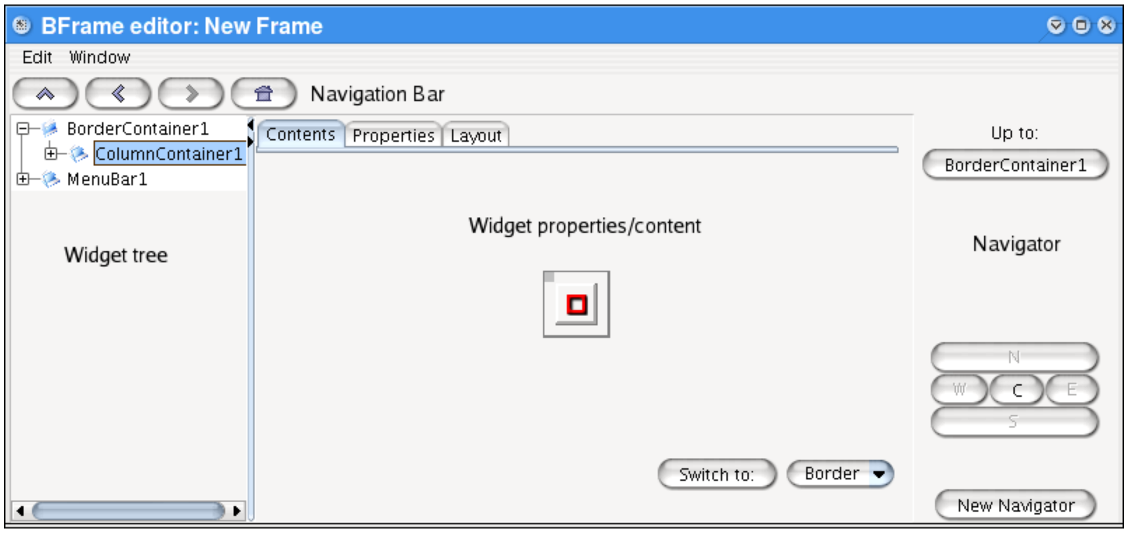


Figure 15. The different areas of the editing window

The Contents edit panel

The figure Figure 16 shows the content panel of a 2 by 2 grid container. There are four thumbnails showing the four 'slots' of the grid. The grid has only one child in the 0,0 element. The thumbnail is green to show that this slot is occupied by a child. Its handle at the top left corner is enabled, meaning that this widget can be dragged around the grid or into the clipboard. All other thumbails show a hollow red sqare which means that they are empty. When an occupied slot is selected, it is surrounded by a green square whereas an empty slot is surrounded by a purple square when selected.

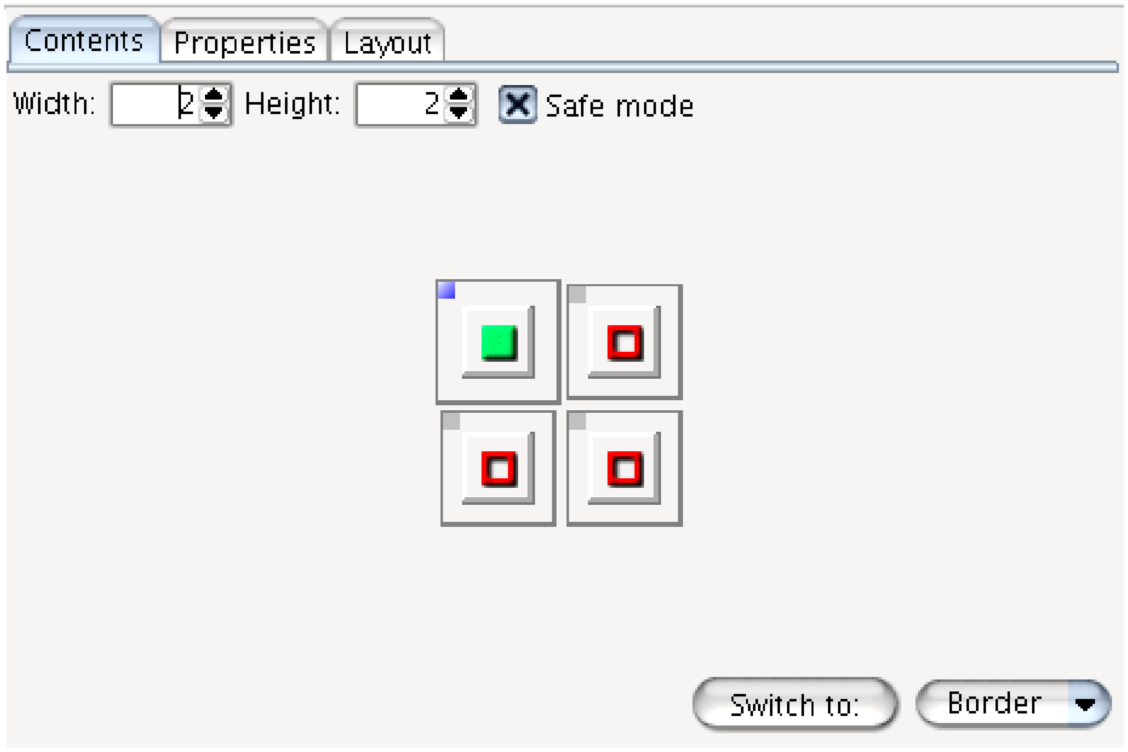


Figure 16. The content panel for a 2 by 2 grid container

There are different ways to add a widget to a container.

1. Select the type of the widget to add in the widget palette. Click on the empty thumbnail where you want to place the new widget. The newly created widget is automatically selected, unless the **Ctrl** key was hold down during operation, in which case the new widget is not selected. This way, several widgets can be successively created.

2. Drag a widget from the clipboard. If the widget is not persistent, it will be erased from the clipboard, unless the **Ctrl** key has been held depressed during operation.

3. Paste (**Ctrl-V**) the currently selected item in the clipboard in the selected empty slot. Selected clipboard item is not discarded even if it is not persistent.

Widgets can be moved around in the container using the blue drag handle. When the target slot is already occupied, behavior slightly depends on the widget container type. For row and column containers, widgets are swapped. For fixed or length specified content container such as border or grid containers, a dialog asks if you want to erase target widget or swap the two widgets.

Widgets can be copied to clipboard as usual, but they can also be dragged to the clipboard. If the **Ctrl** key is held down in the process, a copy is put in the clipboard and the original widget is not deleted.

Sometimes, one wishes to transform a widget container belonging to this group: border, column, row, grid, form, into another container type of the same group, e.g. from grid to form. The content editor displays a button at the top right labeled Switch to. Selecting this button will transform the edited widget into a container of the type specified in the combo box next to the button. Container content is preserved as much as possible.

Some features are specific to each kind of content editor. They are discussed below.

Grid Container and Form Container	The width and height are specified using two spinners at the top left of the panel. When the safe mode check box is enabled, it is not possible to downsize a grid or a form if in doing so a widget is deleted. The user is asked to confirm deletion before setting the new size.
Row and Column Container	There is always an empty slot at the right/ bottom of the container. This slot is virtual and exists only to provide a means to further addition of child widgets to the container.
Form Container	Thumbnails have an orange handle at the bottom right corner. This handle is used to set the horizontal and vertical extension of the widget. The row and columns weights can also be set using the relevant spinners in front (resp. at the top of each row (resp. column).

The Properties edit panel

Since each widget has more or less its own properties, we won't go through the process of describing each property for each widget type. However it is to be noticed that for widget containers that uses a default layout, this layout is listed in the widget properties. Modifying the default layout will affect all child widgets which don't make use of a specific layout.

As for the content edit panel, there are some features which are particular to a specific widget. These are discussed below.

BTree node insertion	BTrees node properties can only be edited when a node is selected in the edited window. For example, to add a node to a BTree, first select the node in the edited window, either through using **Ctrl** click or entering run-time mode. The node properties then become enabled and it is possible to add a node to the selected node.

The Layout edit panel

This panel is used to specify a layout for the selected widget with regards to the parent widget. If no specific layout is given, then the parent widget default layout is used. Do not confuse this specific layout with the default layout setup in the properties edit panel. The default layout affects the widget container child widgets layout whereas the layout specified in this panel affects the way the widget container itself is laid out.

Underneath the layout section, choices specific to the parent container of the edited widget allows to choose the placement of the widget within the parent widget (if relevant). It is however best to use the content panel of the parent widget for this purpose and this feature is mainly for occasional use.

Menus

Edit menu items:

Cut/Copy/Paste/Clear	As usual. The content widget is the only widget which can't be deleted. To change the content widget into anything else than the default border container, use the Switch to: button.
Remove Container, Keep Child	When a widget container is selected, selecting this command removes it from the widget tree. Its first child made child of the widget container parent widget at the place it previously occupied.
Embed in Container	A widget of the kind selected in the widget palette is created and positionned at the place occupied by the selected widget, which in turn is made first child of the newly created container. This command can be used to outline a widget on the fly, for example.
Export Frame as XML...	Exports edited frame as a buoy xml file
Export Selected Widget as XML...	Exports the currently selected widget ias a buoy xml file.
Save as / Save	Saves the edited window into a balise interface definition file. If a project is associated to the file, the frame (or content container if Save = Export content container is checked) is also exported as a buoy interface definition file.
Set Run Time Mode	Default interaction with edited window consists in selecting widgets with the mouse (see section the section called “The Edited Window”). If run time mode is enabled, the edited window acts as a normal window, i.e. it behaves as it does as run time. This mode can be toggled back to edit mode.

Window menu items:

Add a Menu Bar / Remove the Menu Bar	Menu bars cannot be added in a container empty slot. They must be added to a window/dialog using this command. If the edited window already has a menu bar, then this command removes the menu bar from the window.
Window properties	This command triggers a dialog which al-

lows to specify the window title, if the window is resizable and initially visible. For obvious reasons dialogs can't be made modal at design time. It is mandatory to call:

```
setModal(true)
```

at run time *before* making the dialog visible.

Prototype Window Code.../Prototype Widget Code...

This command brings a window in which is displayed the typical code used to load the interface (selected widget or whole window). The code is prototyped according to different options. Variables can be stated as local (on the fly definition) or global (e.g. class scope definition). Variables definition can be stated as `private`, `protected`, etc. Finally, the prototype code can either load the interface from a file that resides on a filesystem (local or network) or from within the jar file from which the class code has been loaded (in which case the code uses the class ClassLoader to load the interface file).

Prototyped code is a feature intended for quick paste into source code. This code is not managed by Balise. Do what you want with it! Do not forget to change the default `xxx.xml` filename to the actual file name.

Explicit Container Grid...

Explicit Containers are the only case where child widgets are placed in the container using the mouse *in the edited window*. A grid can be used to align child widgets. Grid size, Snap to Grid and Show Grid options are set using this command.

Save = Export Content Container

If this item is checked, a Save command does not save the whole window but rather the content container. Useful if you build your windows or dialogs this way:

```
public class AWindow extends BFrame
{
    public AWindow()
    {
        super( "My Window" );
        try
        {
            WidgetDecoder decoder = new Wi
            setContent( (BorderContainer)
        }
        catch ( IOException ex )
        {
            ex.printStackTrace();
        }
        pack();
        setVisible(true);
```

```
          }
      }
```

Project menu items:

New Poject...	Creates a new project file to be associated to the edited interface.
Set Project...	Allows to choose an existing project file to associate to the edited interface.
Edit Project	Allows to change projects parameters, i.e. paths.

The Edited Window

The primary purpose of the edited window is to show the resulting layout of the GUI. Widgets can be selected using the mouse. Parent widgets selection is usually indirect, simply because many parent widget do not display an area in which to click. However, clicking several times in a given widget select its parents. Two clicks in a row select the parent widget, three clicks select the parent of the parent widget, etc.

Widgets cannot be moved around in the edited window. Layout is solely ruled by widget hierarchy, the types of containers used and the LayoutInfos used to lay out the widgets. The only exception to this rule is the ExplicitContainer widget for which child widgets can be placed within the container dragging them with the mouse.

It is sometimes necessary to interact with the widgets themselves rather than just select them. Occasional interaction can be achieved using **Ctrl** clicking. In this case, the **Ctrl** modifier is removed from the event which is forwarded to the widget. This way it is for example possible to interact with a BTree in order to select a node to access its properties in the editing window contents panel. In some cases, the **Ctrl** click is enough, if only because one wants to **Ctrl** click on the widget itself. It is then possible to enter run time mode in which the window behaves as usual. It is then no longer possible to select widgets in the edited window and the widget tree of the editing window must be used.

The Clipboard Window

The clipboard has to purposes:

- Store temporary widgets which have been copied to the clipboard
- Store persistent widgets which will act as templates to be pasted from the clipboard

The term temporary/persistent refers to a Balise session.

Each time a widget is copied, it appears in the clipboard as if it had been dragged there. Every subsequent copy commands create other copies of the selected widget(s) in the clipboard. The last copied widgets is framed in green to show that it is the currently active widget in the clipboard. Every paste command will paste a copy of this active widget. Alternatively, a widget can be dragged from the clipboard to a widget container content panel. It will then be deleted from the clipboard unless the **Ctrl** key has been depressed during operation.

To avoid the clipboard being flooded by copied widgets after a certain time, a maximum number of temporary widgets can be specified. The default is five. This number is a maximum number *per category* (more about categories later on).

In addition to these temporary widgets, there are persistent widgets which are framed in red when selected and whose names are displayed in italic. These widgets are kept from session to session. They are not deleted when dragged from the clipboard to a widget container content. The purpose is to have a stock of persistent template widgets which can be used when needed.

Categories: since it is intended that several persistent widgets may reside in the clipboard, it is possible to spread widgets over categories that are accessed using the tabs of the tabbed pane. After installation, there is only one category named Misc. Categories can be added as needed and widgets sent from one category to another.

Clipboard menu items:

Add a new tab/Remove current tab/Rename current tab	As they command name says. I might rename 'tab' to 'category' in next version.
Delete Widget	Deletes currently selected widget. Please note that multiple selection in the clipboard is not allowed and widgets must be deleted one by one, except when using delete all widgets.
Empty Clipboard	Deletes all *temporary* widgets in the *current category*.
Max Number of Items...	Allows to enter the maximum number of items a categiry can hold. When the clipboard is full, the oldest widget is removed when a new widget is copied (FIFO).

Item menu items:

Set Persistent/Set Temporary	Toggles the persistent character of a widget.
Send Widget to Tab	Allows to send the selected widget to another category.

Balise File & Projects

Since version 1.1, Balise files and data are organized according to projects. This feature is made necessary by the fact that some files (e.g. icon images files) need to be accessed at *edit time* by Balise and at *run time* when the interface is loaded in a totally different context (e.g. jar files). A project defines three directories:

- **The project root folder**. This folder mimics the path structure that will be used at runtime. That is to say that the root subfolder tree at edit time will correspond to the directory structure at run time. Let's suppose a user named john develops an application which is contained in a jar file. The jar file contains two directories, `images` and `interfaces` (one folder for images, one for storing the xml buoy interfaces). Let's also suppose that the root folder for this project is set to `/home/john/application`. The icon images files are stored in `/`

`home/john/application/images` and interface files in `/home/john/application/interfaces`.

- **The images folder**. The purpose of the images folder, as explained in the exemple above, is to place every image used by the interface in location that will be accessed at edit time as well as run time. Whenever an image icon is set (e.g. in a button), the image file is copied into this directory if it does not originate from it. At run time, the `WidgetDecoder` in charge of the interface expects images to be available in `./images`, whether the application is run from a jar file or not.
- **The interfaces folder**. This is the place where Balise exports buoy interface files. In previous versions (1.0x), Balise worked directly on buoy interface files. However, this prevents adding Balise specific information in the interface file. As of 1.1, Balise uses its own file format to store interface definition. Balise files can not directly be used by buoy `WidgetDecoder`. However, each time a balise file is saved, the corresponding buoy interface file is also saved in the interfaces directory. Another benefit from this design is that the prototyping mechanism now knows where the interface is and you no longer need to modify the interface file name in the prototyped code.

Projects are set and edited using the Project menu of the edit window. It is recommended that they use the `.bpr` extension.

Interface definition files are standalone balise files. As compared to buoy interface files, they contain additional information, such as which widget is to be prototyped and which is not. Of course, they also contain the whole buoy interface. If you associate a project file to an interface definition file (which *strongly* recommended), then each time you save the interface, the plain buoy xml definition file will also be exported with the same name in the interfaces folder. An advantage of having buoy interfaces stored in interfaces folder is that the file name is specified in the prototyped code. The same project can be used for several interfaces, but you must have one project each time the folders where you place interfaces and images change. The trade off is that once saved project files and balise files must not be moved. If a balise file is moved, then the project must be edited so that the location of project relative to balise file is correct. If you move a project file, then edit the project to set correctly the root folder and reset the project file location in every balise file that points to this project file.