The Mjølner BETA System
Frigg - User Interface Builder
Tutorial and Reference Manual

Mjølner Informatics Report
MIA 96-33(1.1)
August 1996

Contents

Introduction

Frigg is the interface builder of the Mjølner BETA System, it aims at supporting rapid prototyping and is meant primarily for system designers and developers.

Frigg does not enforce one particular style of development on its users. In fact, Frigg supports at least the following approaches: (1) starting with design of user interface objects (UIOs) and building up a horizontal prototype having only minimal functionality; (2) starting from a model, and an implemented description of the functionality of a system, but with no UIOs implemented; (3) vertical prototyping, i.e. implementing the functionality behind a subset of a horizontal prototype or fully implementing a small subset of a system intended for incremental extension; (4) simulation of functionality, i.e. adding temporary short cut or dummy computations to a horizontal prototype to support sample data; and (5) full application development. In addition, these different ways of using Frigg can be combined.

Frigg is integrated with the other tools of the Mjølner BETA System. It is embedded in Ymer, the Mjølner BETA Sourcebrowser, and supplies a graphical editor which interacts with the structure editor to create a prototyping environment. The graphical editor lets its users construct the user interface via direct manipulation, while the necessary code is generated automatically behind the scene. The generated code can be extended and tailored using the structure editor, and this tailoring can take place at any time during the development of the user interface, because the graphical editor can keep track of the code even though the code has been altered in the structure editor. The developer can therefore alternate between working on the user interface and coding the underlying functionality.

The integration of Frigg's editors is accomplished using the Mjølner BETA System's uniform representation of programs: abstract syntax trees (ASTs). Manipulations of the ASTs are done through the Meta Programming System (MPS). Furthermore, all editors working on the same AST are informed when the AST changes, thereby allowing the editors to ensure consistency. The ASTs generated by Frigg are decorated with special comments that are used by Frigg to recognise user interface objects and to store links to layout information.

Frigg utilises the Mjølner BETA Fragment System to: (1) Get access to Lidskjalv, the Mjølner BETA Graphical User Interface environment (GUIenv), (2) Create user interface modules that are properly separated from the model modules of the program, (3) Separate the interface part and the implementation part of the user interface objects, to allow fine tuning of the users interface without recompiling the entire program.

The code generated by Frigg are specialisation's of the classes in GUIenv. Figure 1 shows a FindDialog created in Frigg, as it appears at runtime.

Figure 1: A FindDialog created in Frigg

The BETA AST that representing the FindDialog is pretty-printed in its textual form below:

```
ORIGIN '~beta/guienv/v1.3.1/guienvall';
BODY 'finddialogbody'

-- guienvLib: Attributes --

findDialog: (*$ 1*) window
  (#
     onFind:< (# str: ^text;  enter str[] do INNER #);
     onCancel:< (#  do INNER #);

     open::<
       (#
       <<SLOT findDialogOpen:DoPart>>
       #);
     private: @<<SLOT findDialogPrivate:Descriptor>>;
  #)
```

The FindDialog fragment group

The first three lines are the fragment syntax, which is what makes the fragment group a GUIenv library that can be used in any GUIenv program.

The FindDialog pattern is a specialisation of the GUIenv window pattern because the FindDialog is a window and therefore inherits from the window class.

The special comment "(*$ 1*)" tells Frigg that this is a user interface pattern created by Frigg. The number 1 serves as a link to layout information stored in a separate data file, created by Frigg. Each GUIenv library managed by Frigg has its own data file for layout information. This is called a resource file.

The interior of the FindDialog is hidden in the implementation file called a BODY file. This allows the developer to change the content of the FindDialog without recompiling the part of the program which uses the dialog. The user interface objects in the window are declared in the private part of the FindDialog pattern as individual singular objects.

The parts of the FindDialog that are automatically generated is the further binding of "`open`" and the "`private: @<<...>>`" declaration. The two virtual procedures "`OnCancel`" and "`OnFind`" are added by the developer as the interface between the FindDialog and the application. The "OnFind" virtual procedure is called when the user presses the "find" button in the dialog. The argument to "OnFind" is the contents of the text field in the dialog. Below is a pretty-print of the AST representing the find-button in the private part of the dialog:

```
findBtn: (*$ 7*) @pushButton
  (#
     open::< (# do (* initialize *) #);
     eventHandler::<
       (# onMouseUp::< (# do searchFld.contents -> onFind;  #) #)
  #)
```

The find button of the dialog.

The only part of the FindBtn singular object which is added by the developer, is the dopart of the OnMouseUp virtual. The OnMouseUp virtual procedure is automatically called by the GUIenv system, when the user presses the button. Here the developer has programmed the button to call the "OnFind" virtual procedure in the interface of the FindDialog.

Reference Manual

Interface builder Menu

The interface builder is used from within the Sourcebrowser (See the Sif manual). What is seen when Frigg is started is the sourcebrowser with an extra menu, the interface builder menu. In figure 2 is shown the main sourcebrowser window with the interface builder menu pulled down.

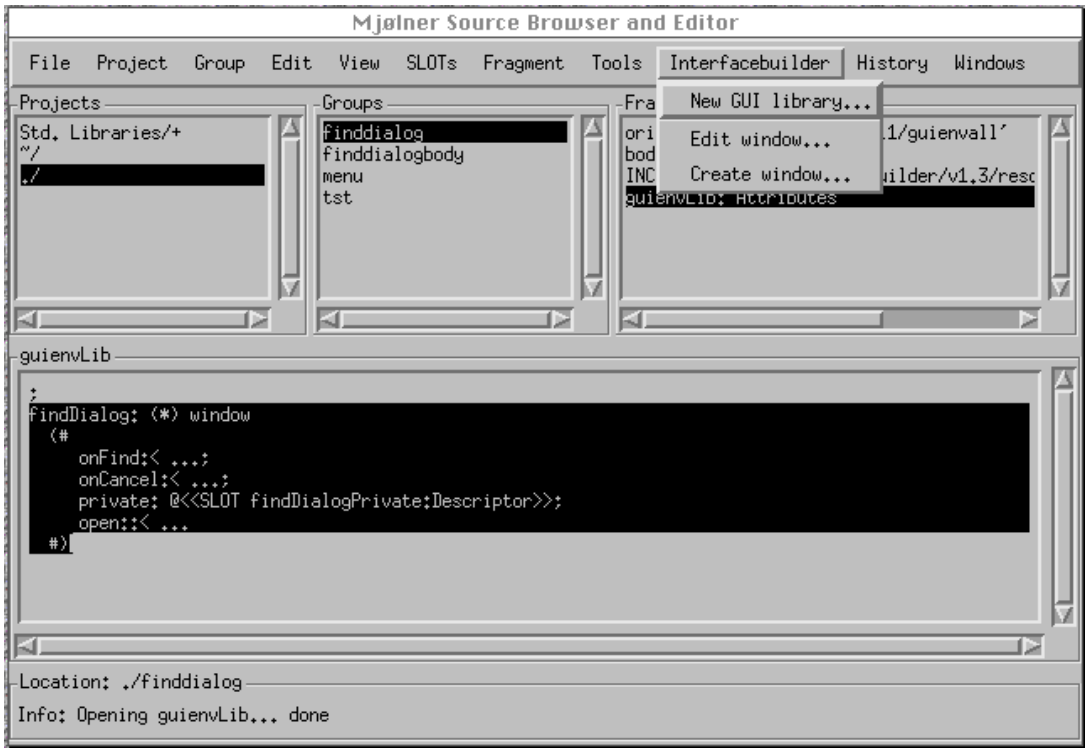


Figure 2: The sourcebrowser with the interface builder menu pulled down

New GUI library

This operation creates a fragment group which is ready to contain user interface code. Therefore the fragment group contains a GUIenvLib fragment which is where the window declarations go, and the ORIGIN is set to guienvall. A body file is also created as well as a resource file.

A filedialog prompts for a filename which will be the name of the new fragment group.

Edit Window

Selecting "edit window" will open a graphical editor on the window pattern that are currently selected in the structure-editor.

Create Window

A specialisation of window is generated in the currently selected GUIenvLib fragment and a graphical editor is opened on the newly created window.

Graphical Editor

The graphical editor is the part of interface builder which allows the desired user interface to be constructed interactively by direct manipulation of Lidskjalv objects. Figure 2 below shows the graphical editor opened on the FindDialog which was discussed in the introduction.

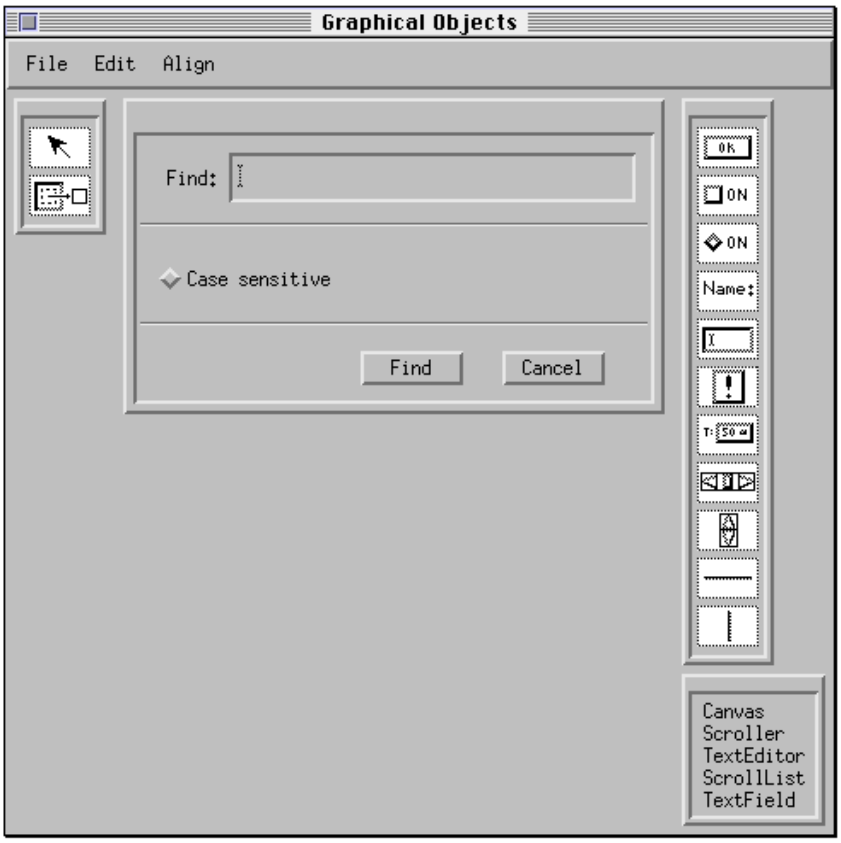


Figure 2: The graphical editor.

To the left the selection tools is found. In the middle the window being edited is shown - in this case the FindDialog. To the right two object palettes are located. The topmost is iconic and presents the array of simple objects available in Lidskjalv, whereas the bottommost is textual and presents the aggregate Lidskjalv objects. The placements of these items in figure 2 is the default configuration. The palettes and the edited window can freely be repositioned .

Selection Tools

There are two selection tools: the arrow tool and the hierarchy tool. Both tools can be used for making selections. A selection is a temporary grouping of objects that can be manipulated as a whole. The selected objects are marked with a special border. An object is selected simply by clicking on it. A shift-click (holding down the shift-key while clicking) on an object will add it to the selection if it was not selected and remove it from the selection otherwise. A simple click on an unsolicited object will make that object the only object in the selection.

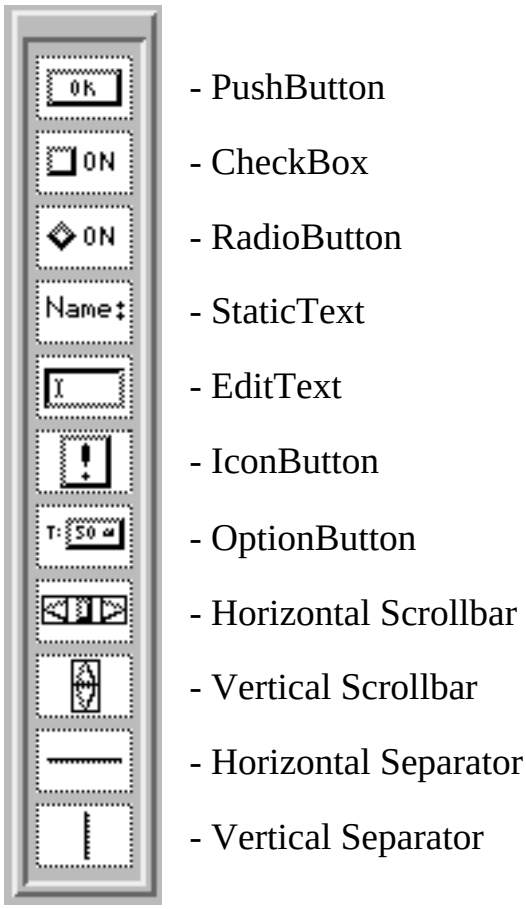
The arrow tool is used for moving and resizing the objects in the window. If the mouse is pressed near the edge inside the object, it is a resize - otherwise it is a move. When moving objects, they stay inside the canvas they were placed in. If the object by accident is moved but a resize was intended - or vice versa, the action can simply be undone by invoking the "Undo" command in the edit menu.

The hierarchy tool is used to move a selection from one canvas to another inside the window in order to change the hierarchy of the objects in the selection. The graphical aggregation hierarchy of the objects in the window corresponds to the aggregation hierarchy in the underlying beta code, therefore a change in the object hierarchy will be accompanied by a similar change in the beta code.

A tool is selected by clicking the icon - the default selected tool is the arrow tool There is always one of the two selection tools selected.

Object Palettes

Objects are added to the window by utilising the two object palettes. An instance of some object on a palette is created by dragging the object and placing it in the window. When an object is dragged from the palette, the receiving canvas is designated by the mouse, and the border of the canvas is highlighted to indicate where the object will be dropped. Figure 3 explains which pattern in Lidskjalv each item on the simple object palette corresponds to.



- PushButton
- CheckBox
- RadioButton
- StaticText
- EditText
- IconButton
- OptionButton
- Horizontal Scrollbar
- Vertical Scrollbar
- Horizontal Separator
- Vertical Separator

Figure 3: The simple object palette.

The items on the aggregate object palette correspond to the patterns in Lidskjalv with the same name.

File Menu

The file menu is shown in figure 4 below.

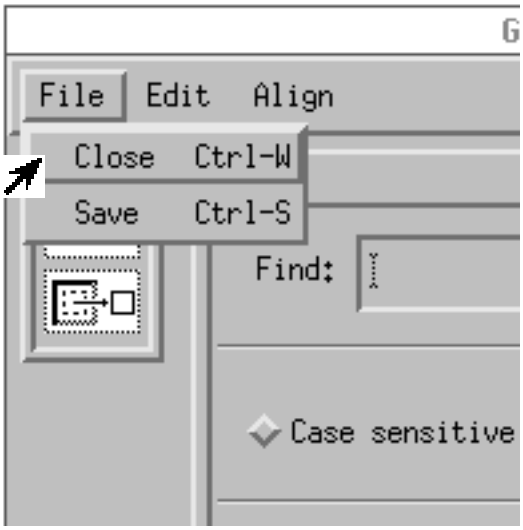


Figure 4: The file menu

Close

Closes the graphical editor after asking the user to save any unsaved changes. It is possible to discard changes by clicking the No button in the dialog.

Save

Save any changes to disk. This operation saves both the changes to the layout and the changes to the code associated with the window. Some changes only affect the private part of the window and other changes affect the interface part and again other changes affect the layout only. The save operation only saves what actually needs to be saved. This means that some layout aspects of the window can be changed without needing to recompile any part of the project and the implementation of the window can be changed without needing to recompile anything but the body file of the window.

Edit Menu

The edit menu is shown in figure 5.

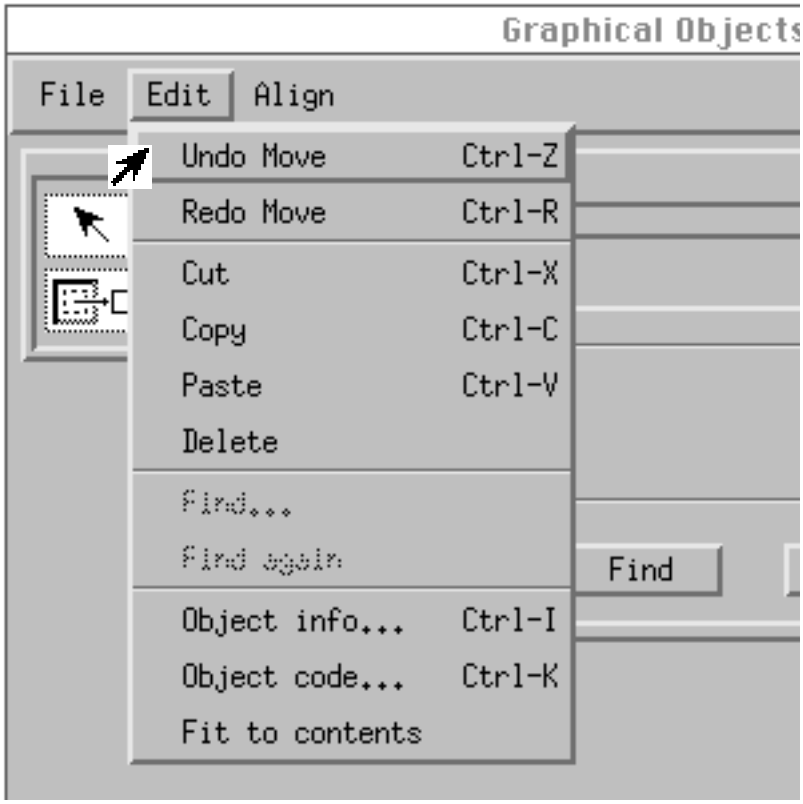


Figure 5: The edit menu.

Undo

All the operations in the graphical editor can be undone by choosing the "Undo" command in the edit menu. The undo is multilevel, which means that all changes can be undone all the way back to when the window was opened for editing in the session.

Redo

A sequence of undo-commands can be redone by invoking the redo command in the edit menu - as long as no other operation has been performed after the undoing.

Cut

A copy of the selected objects is placed on the clipboard along with the underlying BETA code, and then deleted from the window. The object scan the be pasted into the window again. It is possible to copy and paste between graphical editors.

Copy

A copy of the selected objects is placed on the clipboard along with the underlying BETA code. The objects can the be pasted into the window again. It is possible to copy and paste between graphical editors.

Paste

The content of the clipboard is placed in the window at the top level and the underlying BETA code is inserted in the appropriate place. The operation can be repeated to obtain as many copies as needed.

Delete

The selected objects along with the underlying BETA code are deleted without affecting the clipboard.

Object Info

The object info dialog in the edit menu allows the layout properties of the objects in the window to be edited in a dialog form in contrast to direct manipulation. Some properties can be changed in the dialog that cannot be changed otherwise.

Object Code

The object code command gives access to view and edit the underlying BETA code. Later in this chapter, a detailed description of the object code dialog will be found.

Fit to Contents

Frequently, there is a natural size of an object, dependent of the content of the object. For example a StaticText object would have the extent of the text as the natural size. The "Fit to Contents" command in the edit menu will adjust the size of the selected object to its natural size. Keep in mind that not all objects have natural sizes.

Align Menu

The alignment commands in the alignment menu supply facilities to align objects in a row or centred underneath each other etc. Figure 6 below displays the align menu.

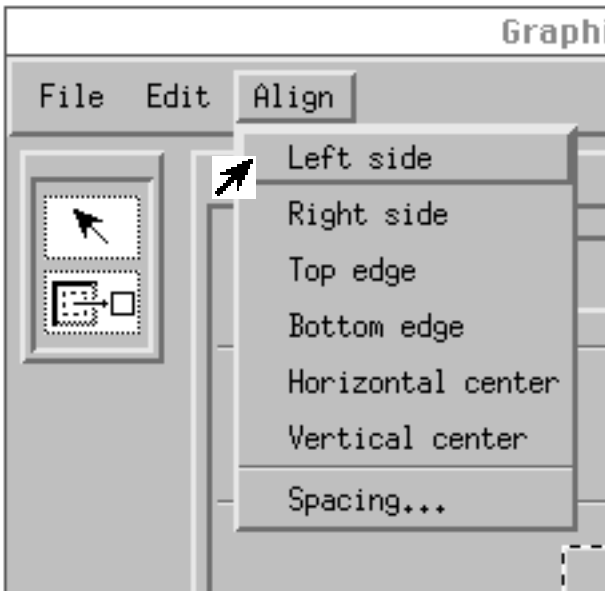


Figure 6: The align menu.

The alignment commands work on the current selection. The first object selected will stay where it is.

These alignment commands are available:

Align left side

Aligns the left sides of the selected objects to the first selected object.

Align right side

Aligns the right sides of the selected objects to the first selected object.

Align top edge

Aligns the top edges of the selected objects to the first selected object.

Align bottom edge

Aligns the bottom edges of the selected objects to the first selected object.

Align vertical centre

Aligns the vertical centres of the selected objects to the first selected object.

Align horizontal centre

Aligns the horizontal centres of the selected objects to the first selected object.

Spacing

This command will present a dialog that allows a group of objects to be given the same size or the objects to be arranged, so the distance between any two adjacent objects is the same. Below there is a screen shot of the dialog:

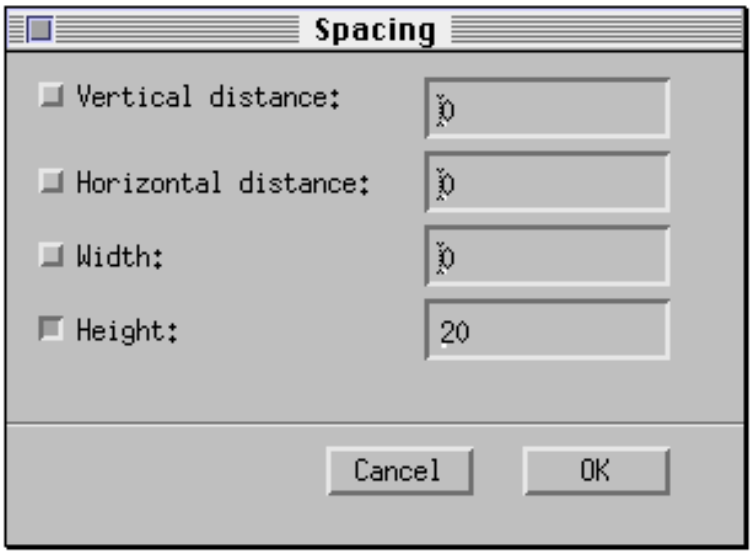


Figure 7: The spacing dialog.

To give a group of objects the same height do the following:

1) Select the objects that should be given the same height
2) Choose the Spacing... command
3) Check the "Height" CheckBox and make sure the other check boxes are unchecked
4) Type the desired height into the field next to the check box
5) Press OK

NOTE: The objects will keep their vertical and horizontal order in the window when adjusting the distance. Furthermore, the top left object will always stay where it is.

Object Info Dialog

Below is a picture of the ObjectInfo dialog for the "find" pushbutton in the FindDialog.

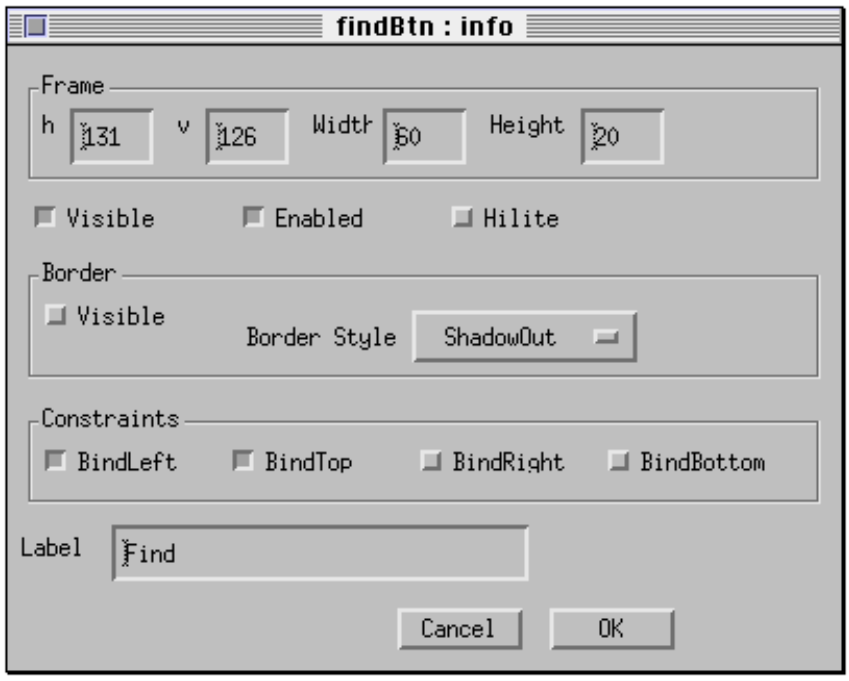


Figure 8: The object info dialog.

The "frame" pane allows the position and size of the button to be typed as an alternative to direct manipulation.

The object can be hidden by unchecking the "visible" radiobutton and deactivated by unchecking the "enabled" radiobutton.

The "border" pane controls the border characteristics. The object is given a border by checking the "visible" inside the "border" pane. The popup menu presents a various border styles.

The "constraints" pane controls how the object reacts when the surrounding object is resized.

Finally the label of the pushbutton can be changed by typing in a new text in the "label" field.

Object Code Dialog

The object code dialog is the interface to the underlying beta code. Figure 9 shows the object code dialog for the find button.

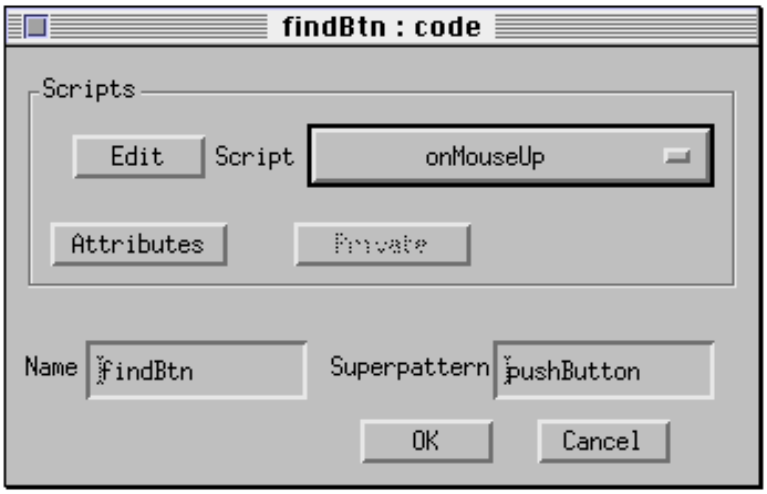


Figure 9: The object code dialog.

The scripts pane gives access to extend the code of the pushbutton. The popupmenu inside the scripts pane shows the available event patterns that can be further bound for this kind of object. Press the "edit" button to get a structure editor on the event pattern currently selected in the popup menu. If the event pattern is not already further bound in the code, a further binding declaration is inserted in the eventhandler of the object.

The "attributes" button opens a structure editor on all the public attributes of the object, and the "private" button opens a structure editor on the private attributes of the object.

To change the name used for the object in the code, type in a new name in the "name" field.

Using the code outside Frigg

Utilising the windows

To utilise a window in a project, the fragment group created by Frigg, in which the window resides, must by included in the fragment group, where the window are needed.

An instance of the window can then be created an opened.

Below is a simple example using the FindDialog:

```
ORIGIN '~beta/guienv/v1.3.1/guienv';
INCLUDE 'finddialog';
-- program: Descriptor --
GUIenv
  (#
     theFindDialog: @findDialog
       (# onFind:: (#  do str[]->putLine;  #);  #);

  do theFindDialog.open
  #)
```

Using the FindDialog

Using Other Editors

The fragment groups are saved on disk in .ast files which contains the abstract syntax trees in a binary data format, but Along with the .ast file there is also a .bet file which is a textual version. The .bet file can be edited in a normal text editor and still be parsed and understood by Frigg.

When editing the fragment groups created by Frigg in the structure editor or in a text editor, the aggregation structure of the windows must not be changed. The special comments (*$ <n>*) should also be left unaltered. Otherwise the window can not be edited in the graphical editor anymore.