Quick Guide to the Gaffe Designer

Nicholas Daley
Email: ntdaley@acm.org

March 15, 2004

What is Gaffe

Gaffe is a customisable GUI front-end to a Z animator.

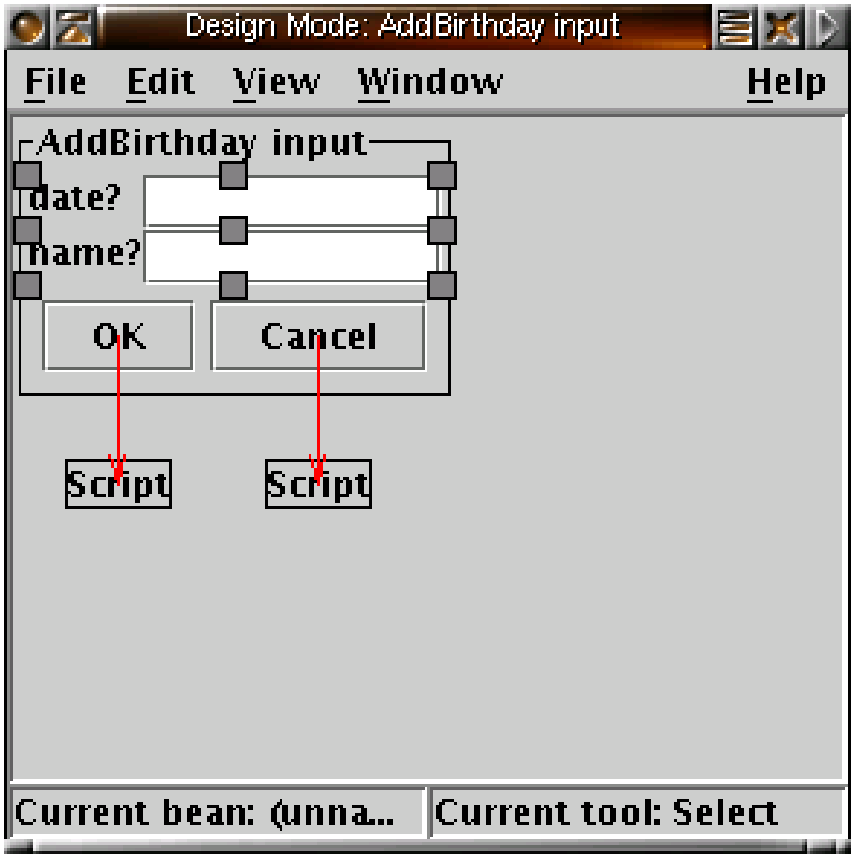
Z is a language for writing specifications for systems, including software systems. Using Z, properties of the system can be tested before it is created, ambiguities and contradictions can be found, through refinement program code can be created, and test-sets can be generated.

A Z animator is a program that allows you to test the result of a sequence of operations described by a Z specification; to a certain extent, emulating the behaviour of the final system.

Unfortunately to interpret a Z specification, or to use a Z animator users often need experience with Z. Gaffe attempts to solve this, by presenting a friendlier graphical interface to the animator. As a result, a client who doesn't know Z will, hopefully, be able to test a Z specification to make sure it matches what is wanted.

Gaffe is split into three programs. The Gaffe generator automatically creates a custom user interface to use with a Z specification provided to it. The Gaffe designer provides a graphical interface builder for creating or modifying the custom user interface. The Gaffe animator attaches to the Z animator, and allows the custom interfaces created with the generator and designer to be used to test a Z specification.

The Form Design Window



The form design window is where you edit one window, or 'form', of the Gaffe interface.

The menus are described in the section 'Menus'. The status bar, at the bottom of the window, is divided into two sections; one describes the currently selected bean (showing its 'name' property, and its type), the other shows the currently selected tool (See the section on the toolbox). If you can't see all of the text in the status bar, making the window larger will reveal the missing text.

Beans

The form itself is represented in the form design window as a bordered rectangle titled by the form's 'name' property (In the example above, this is 'AddBirthday input'). JavaBeans make up the remaining objects in the interface. A bean is a type of object that is usable in application builders. Beans can be visual-beans (or components)[1], for example buttons, windows, or labels; or they can be non-visual and control behaviour. The form design window will not allow non-visual beans to be placed inside the form, and will not allow components to be placed outside the form.

Events and Listeners

Some beans are able to listen for events triggered by other beans. These beans, imaginitively enough, are called 'listeners'. You can register a listener with a bean using the 'Event Link' tool. By default the form design window does not show you the event links connecting beans. The 'Highlight Event Links' submenu of the 'View' menu allows you to show some or all of these event links. Event Links are displayed as a coloured arrow pointing from one bean to another. To identify what type of event the link is for, rest the mouse over the link, and a tool-tip will appear displaying the associated listener type. In the example above each button can trigger an ActionEvent which is listened for by the respective scripts.

Handles

The small grey boxes visible in the example above are handles for resizing, or moving the currently selected bean. If the component containing the current bean has a layout manager associated with it, then it may force a certain location, or size on a bean.

Menus

File

- New Form:
  Creates a new form which opens in a seperate form design window.
- Delete Form:
  Deletes the form in this window, closing the window.
- Save...:
  Save the interface to a `.gaffe` file.

[1]It should be noted that components can be invisible, for example a label with no text in will not be visible.

- Open...:
  Loads the interface stored in a `.gaffe` file, losing the interface currently being edited.

- Import...:
  Loads the interface stored in a `.gaffe` file, adding all of its forms to the ones currently being edited.

- Quit...:
  Exit from Gaffe.

View

- Highlight Beans:
  Draw an outline around all, or some beans in the interface. (Handy if you've created a label with no text in, and now you can't find it.

- Highlight Event Links:
  Show some or all of the event links between beans as arrows. (You'll need to turn this on if you want to see which beans are connected to which listeners).

Window

- Show Properties Window:
  Show the Properties Window, where you can see and edit the properties stored inside a bean.

- Show Toolbox Window:
  Show the Toolbox Window, where you can select tools to perform different operations on the interface.

- Init Script...:
  Edit the initialisation script for the specification, so that global variables and functions can be declared for the interface.

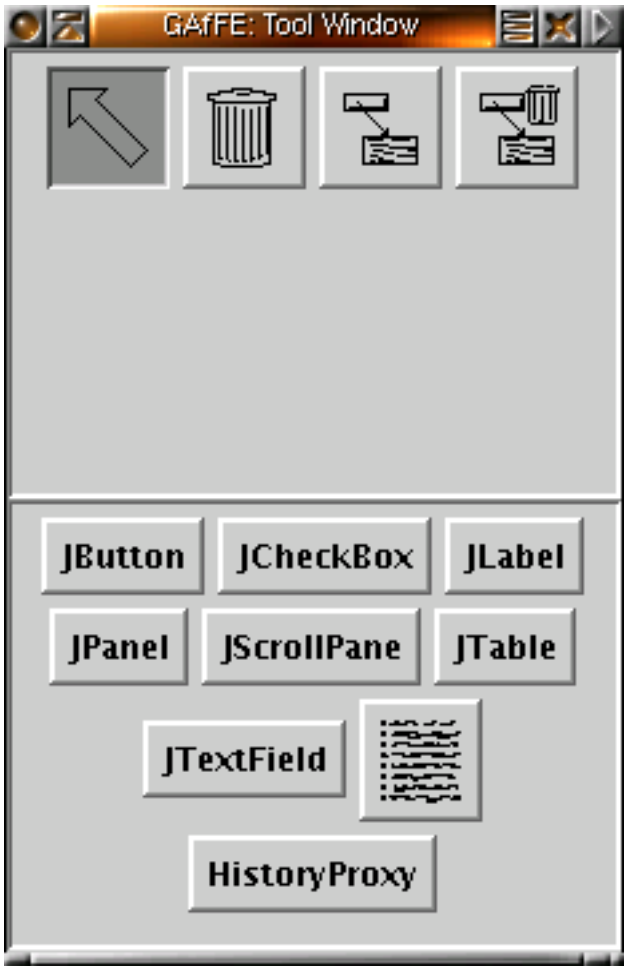
- *The Form Design Windows:*
  Each form design window can be shown using its entry in the Window menu.

Help

- About...:
  The standard about dialog with copyright notice.

- License...:
  Displays a copy of Gaffe's License - the GPL (GNU General Public License).

The Toolbox



The toolbox window provides tools to use in the form design windows. It is split into two panes; tools for creating new beans are in the bottom pane, the remainder are in the top pane.

It is possible to tell which tool is currently activated in two ways; you can look at the status bar of the form design window, or the button for the current tool appears to be pushed down in the tool window.

Select


The select tool allows you to select the current bean. By clicking inside the current bean, you can select one of its descendants.

You can also change which bean is selected through the keyboard: `[Tab]` will cycle from the current bean to one of its siblings in the object containing it. `[Space]` allows you to descend from the current bean to its descendants.

After each tool is used, it will automatically return to the select tool.

Delete


The delete tool deletes the current bean. This can also be done using the `[Delete]` key on the keyboard.

Event Link


The event link tool causes one bean to listen for events from another. When this tool is selected the mouse cursor will change to indicate which beans can have listeners registered with them. To create a link between two beans click on the bean that will trigger the event (e.g. a button), and drag to the listener bean that will respond to that event (e.g. a script).

Delete Event Link


To delete an event link from bean to listener, select this tool, then click on the arrow representing the event link.

The Visual Beans

The names on these buttons will be familiar to programmers who have used Java's Swing toolkit. JButton is a button similar to 'OK' or 'Cancel' from the example in the form design section. JCheckBox is a checkbox. JLabel displays non-editable text (e.g. 'date?' in the example). JPanel allows you to group components together by placing them in a panel together. JScrollPane is another container which adds scrollbars to its contents. JTable provides a table (JTable should generally be placed inside a JScrollPane). JTextField is a field where text can be entered/edited (e.g. The two white boxes in the example).

Script

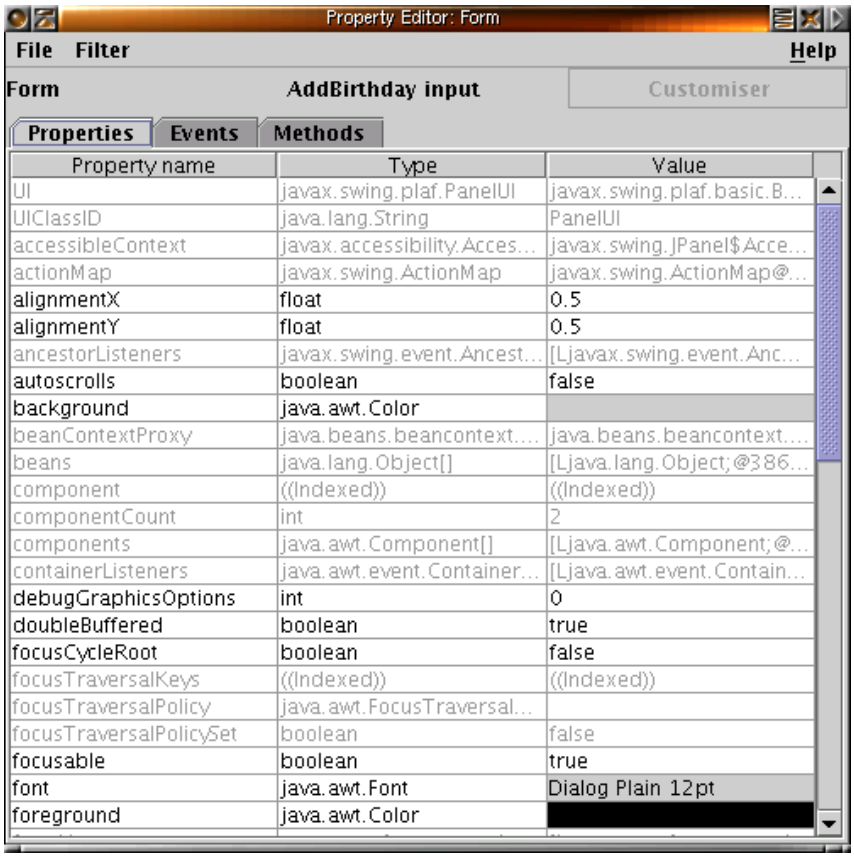

The script bean is a listener that when triggered by an event will run the script stored in its properties (See the section on the properties table). This is the easiest way to customise the behaviour of a Gaffe interface without writing new beans.

HistoryProxy

The History Proxy forwards events from the Gaffe Animator's History.

In the Gaffe animator, whenever the current solution changes, the history sends events to all of its listeners. The History Proxy forwards these events. So by attaching listeners to it, you can cause those listeners to be activated every time the user steps through the history, or activates a Z schema. In the case of the BirthdayBook example, the History Proxy triggers scripts that update the values displayed in the state window and the output windows.

The Properties Window



The properties window is the place where you can edit the state of a bean. For a 'Script' bean this includes the text of the script, for a button this includes the text to display in it, and the colour and font to use. Most beans provide a 'name' property which is mostly useful for scripts to identify beans.

The greater part of the window is made up of tables in three tabbed panes. One shows the methods provided by the bean, one shows the events the bean can trigger, and the third and most useful shows the properties of the bean.

Above this table the name, and type of the bean is displayed, and (if one is available) a button to activate a customiser wizard for the bean.

The properties table shows the name of each property, its type, and its value. Properties that are not greyed out are editable by clicking in the value column on that property's row.

The filter menu on this window allows you to cut down the number of properties shown, to make using the table easier.

Writing Scripts

Although it is possible to configure Gaffe to use other scripting languages, the default is to use ECMAScript (better known as JavaScript).

Three objects are made globally available to all scripts:

- History: Used for stepping through the history, and for activating new schemas.
- AnimatorCore: The main object of the Gaffe animator.
- Forms: The list of all of the forms in the Gaffe interface.

History

History typically provides the following methods and properties:

- `ZBinding currentSolution`: The current solution to be displayed.
- `String positionLabel`: A label that can be used to display the current location in the history.
- `void nextSolution()`: Goes to the next solution in the current point in the history.
- `void previousSolution()`: Goes to the previous solution in the current point in the history.
- `void nextSolutionSet()`: Goes to the next place in the history.
- `void previousSolutionSet()`: Goes to the previous place in the history.
- `void addInput(...)`: A number of methods for adding input, but it's easier to use the `fillHistory` function.
- `void activateSchema(String schemaName)`: Runs a schema through the animator using the inputs given with `addInput`.

Several ECMAScript functions are available in files in Gaffe's JAR file.

- `void fillHistory(Form form)`: Finds text components (e.g. text fields) in the given form whose names match variables in the currently displayed solution. It then copies the data from the component into the History using `History.addInput`. So to activate a schema, all that is needed is to run `fillHistory` passing it the input forms, and then run `History.activateSchema`.

- `void fillBeans(Form form)`: Finds components in the given form whose names match variables in the currently displayed solution. If the component is a supported type (A text component or label, or a JTable with one of the table models that come with Gaffe), then it copies the data for the variable into the component so that it will be displayed.
- `void clearBeans(Form form)`: Clears any components that would be filled by `fillBeans` or read by `fillHistory`.