shortcut

Your Short Cut to Knowledge

ThinWire® Handbook

A Guide to Creating Effective Ajax Applications

Joshua Gertzen and Ted C. Howard

SOURCEFORGE®

Community Press

www.prenhallprofessional.com

ThinWire

Open Source Ajax RIA Framework

Visit us on the Web: www.prenhallprofessional.com

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
United States of America
Fax (617) 671-3447

ISBN-13: 978-0-13-2366229
ISBN-10: 0-13-236622-3

Second release, August, 2007

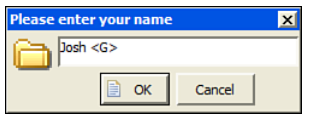

FIGURE 2.10 Running the AllAboutMessageBox example.

LISTING 2.10 Continued

```
} else {
      //Picked mabye, so suggest the user has a problem
      MessageBox.confirm("What? Do you have amnesia?");
}

//Closes the nonblocking MessageBox
MessageBox.closeCurrent();
```

MessageBox Image Locations

For either the icon or button images that `MessageBox` supports, you can specify the location of the images using any of supported resource locator formats that ThinWire supports throughout the framework (for example, the `class:///` or `http://` syntax or even an actual server file location). The final chapter has a small section describing this, so refer to that for more details.

FileChooser

Uploading files and processing them in Web applications has long been a bit of a pain in the neck. To top it off, the only way to send a file to the server is by performing a full form submission using a specific HTML input control. Because this obviously does not gel with the ThinWire philosophy, we spent numerous hours trying to develop a workaround for this issue. Part of what made it so difficult is the lofty goal we had of using an actual ThinWire `Button` and `TextField` in place of the standard file upload control of HTML. In the end, after much back-breaking work, it all works perfectly. This means that our `FileChooser Component` can do something that no other can; you can style ours just like you would any other `Component` in the framework! And you, my friend, get to indulge in the simplicity and ease of use it provides.

The `FileChooser` can actually be used in a number of different configurations. The first and most integrated approach is to just create an instance of `FileChooser` and add it directly to a `Container` within your application. It should blend in perfectly with the rest of your application if you do it this way, because you have complete control over its placement and size. To get things running more quickly, you can alternatively use one of the static `FileChooser.show` methods to display a `Dialog` in one of a few configurations. The `FileChooserMania` example in Listing 2.11 demonstrates both the integrated use of `FileChooser` and the `Dialog` form (see Figure 2.11).

LISTING 2.11 Using the FileChooser as a Component and Compound Dialog

```
//Retrieve the browser frame from the application object
Frame canvas = Application.current().getFrame();

//Create the file chooser component, add it like normal
final FileChooser fc = new FileChooser();
canvas.getChildren().add(fc.setBounds(10, 10, 200, 25));

//An Upload Button
Button btn = new Button("Upload");
canvas.getChildren().add(btn.setBounds(10, 40, 90, 25));
btn.addActionListener("click", new ActionListener() {
      public void actionPerformed(ActionEvent ev) {
            //Triggers actual file upload to occur
            FileChooser.FileInfo fi = fc.getFileInfo();

            //Ask if we should save
            int btn = MessageBox.confirm(null, "File Upload:" +
                  fi.getName(), "Save to Disk?", "Yes|No");

            //Save to base folder of application on server
            if (btn == 0) fi.saveToFile(fi.getName());
}
});

//An Upload Dialog Launcher
```

LISTING 2.11 Continued

```
btn = new Button("Upload Dialog");
canvas.getChildren().add(btn.setBounds(110, 40, 100, 25));
btn.addActionListener("click", new ActionListener() {
      public void actionPerformed(ActionEvent ev) {
            //Ask what type of upload dialog
            int btn = MessageBox.confirm(null, "Show Upload Dialog",
                        "What type to show?",
                        "Single|SingleDesc|MultiDesc");

            if (btn == 0) {
                  //Upload one file
                  FileChooser.FileInfo fi = FileChooser.show();
            } else if (btn == 1) {
                  //Upload one file, with description
                  FileChooser.FileInfo fi = FileChooser.show(true);
            } else {
                  //Upload multiple files with descriptions
                  List<FileChooser.FileInfo> files = FileChooser
.show(true, true);
            }
      }
});
```

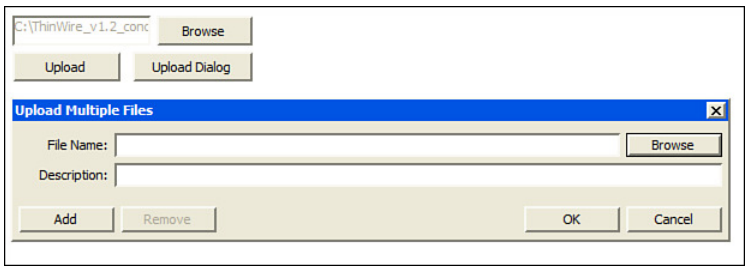

FIGURE 2.11 Running the FileChooserMania example and pulling up the multi-upload dialog.

Visualize Data with GridBox

One of the most powerful and useful `Components` in the framework is the `GridBox Component`. This `Component` enables you to visually display data to a user, while at the same time allowing them to interact with it by sorting, resizing columns, selecting rows, and in some cases, checking rows. A `GridBox` is primarily made up of two `java.util.List` objects. The first is accessible by calling `getRows` and may only contain `GridBox.Row` instances. The second is accessible by calling `getColumns` and may only contain `GridBox.Column`. Listing 2.12 outlines the usage of some of the more common features.

LISTING 2.12 Using Advanced Display Format and Comparator Features

```
//Create a GridBox with visible column headers and check boxes
GridBox gb = new GridBox();
gb.setVisibleHeader(true);
gb.setVisibleCheckBoxes(true);
gb.getColumns().add(new GridBox.Column("Name", true, 100));
GridBox.Column priceCol = new GridBox.Column("Price", true, 50);
gb.getColumns().add(priceCol);
gb.getColumns().add(new GridBox.Column("Definition", true, 200));
```


LISTING 2.12 Continued

```
//Special formatter to make numbers pretty
priceCol.setDisplayFormat(new GridBox.Column.Format() {
      public Object format(Object obj) {
            return "$" + obj + ".00";
      }
});

//Numeric Comparator Guaranatees Proper Sort
priceCol.setSortComparator(new Comparator() {
      public int compare(Object o1, Object o2) {
            return ((Integer)o1).compareTo((Integer)o2);
      }
});

//Populate the grid with rows of Fruit!
for (Fruit f : getFruits()) {
      GridBox.Row row = new GridBox.Row();
      row.add(f.getName());
      row.add(f.getPrice());
      row.add(f.getDefinition());
      gb.getRows().add(row);
}

//Retrieve the browser frame and add the Tree
Frame canvas = Application.current().getFrame();
canvas.getChildren().add(gb.setBounds(5, 5, 400, 200));
```

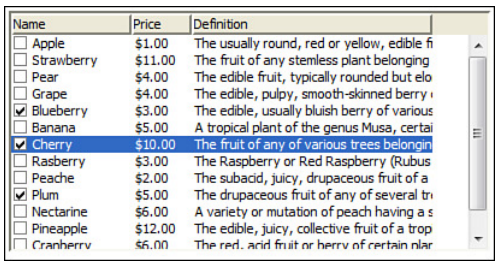

FIGURE 2.12
Running the FruitGrid example and checking some fruit rows.

Sorting a GridBox from Code

You already know that a user can sort a column by clicking a column header. What you probably didn't know, however, is that you can cause a sort to occur directly in your code by invoking `setSortOrder` on a column and passing it one of the constants from `GridBox.Column.SortOrder`, such as `ASC` for ascending sort.

The constructor for `Column` specifies the name of the column, whether it is visible, and in this case, the width of the column. A number of other overloaded constructors enable you to do you a range of tasks. The contents of the `GridBox` may be sorted based on any column. The user needs only to click the column header, and the contents will be sorted ascending. If the user clicks the column again, it will sort descending. This behavior is enabled by default but can be disabled by calling `setSortAllowed(false)`. The sorting of the column is performed by a `Comparator`. By default, each column has an alphabetic `Comparator`. If one of the columns contains numeric values, you'll want to change the sort `Comparator` like we do in Listing 2.12 so that the sorting works properly.

Another interesting thing you can do is use a `GridBox.Column.Format` instance to translate the display values for a column from one format to another. By default, you can stick any type of object into a `GridBox` row or column, but during rendering, the `toString` method on the object will be called to get its text value. By creating a `Format`, you can overwrite this behavior and determine what you want to actually be displayed for a column.

