Extending FEST-Swing

It is very likely that, in your application, you are using "custom" Swing components (e.g. Flamingo, JIDE, or your own.) Since FEST-Swing currently supports only the "standard" Swing components (the ones that come in the JDK,) you might want to create your own FEST fixtures to test your application.

The following are some suggestions or recommendations you can follow when creating your own FEST fixture:

1. Take a look at the code and Javadocs of existing FEST fixtures

By reading FEST's code you can learn to use the `BasicRobot` to simulate a user moving the mouse, clicking a mouse button or pressing keyboard keys. We have separated the structure of a component fixture in several layers (from bottom to top):

1. `BasicRobot`. Simulates a user interacting with a mouse and keyboard. It uses the AWT Robot to generate native input events.
2. Component driver. This layer does all the "heavy lifting." All interaction with a GUI component is done in this layer. It knows how to simulate events and check the state of a specific GUI component. For example, `JComboBoxDriver` knows how to simulate a user using a `JComboBox` (selecting a particular element) and how to verify the state of it (which element should be selected.)
3. Component fixture. This layer sits on top of the driver. It provides a fluent interface to that makes the API easier to write and read. Users of FEST write their GUI tests using fixtures, not drivers. Fixtures can be considered the "user interface" of the FEST-Swing library.

Javadoc documentation can be found here.

2. Extend an existing FEST fixture

If the component you want to test is a subclass of a JDK Swing component (e.g. you have a custom button that extends `JButton`) you can extend an existing concrete FEST fixture (`JButtonFixture` in our example.)

If the custom GUI component does not extend any JDK Swig component, or if you prefer to create a FEST fixture from scratch, please read the following:

- Extend `ComponentFixture`. This class provides all the necessary wiring of a GUI component to test and a Robot. It also provides some very basic functionality and convenience methods.
- If want to simulate a user right-clicking on your component and showing a `JPopupMenu`, extend `JPopupMenuInvokerFixture`.
- Implement any of the following interfaces if needed:

1. `CommonComponentFixture`
2. `FocusableComponentFixture`
3. `KeyboardInputSimulationFixture`
4. `MouseInputSimulationFixture`
5. `StateVerificationFixture`

- You can also extend `GenericComponentFixture`, which extends `ComponentFixture` and implements the interfaces mentioned in the previous point.

> **Warning**
> To avoid unexpected side effects in your tests, you **must** access Swing components in the event dispatch thread.

3. Create an extension for ContainerFixture

By default, implementations of `org.fest.swing.fixture.ContainerFixture` provide shortcut methods to access the standard Swing components (the ones that come in the JDK) in a `Container`. For example, the following code listing shows shortcuts methods to access a `JLabel` and a `JTree` from a `JFrame` being managed by a `FrameFixture`:

```
FrameFixture frame = new FrameFixture(new MyFrame());
frame.label("pathLabel").requireText("Path:");
frame.tree("navigationTree").selectPath("c:/projects/fest");
```

If you have created a fixture for your custom GUI component, it is not possible to add a shortcut to `Container` due to the lack of extension methods in Java. For example, let's assume you have created a custom GUI component called `MyCalendar` and a fixture to use this custom component in your GUI tests called `MyCalendarFixture`. It is not possible to add the shortcut method `calendar` to `ContainerFixture` and have all its implementations look like this:

```
frame.calendar("myCalendar").selectDate("10/18/08");
```

To overcome this limitation, FEST-Swing provides `ComponentFixtureExtension`. The following code listing shows an extension to add a shortcut to a `MyCalendarFixture`. This extension looks up a `MyCalendar` that has a matching name and is showing on the screen:

```
public class MyCalendarFixtureExtension extends ComponentFixtureExtension<MyCalendar,
MyCalendarFixture> {

  public static MyCalendarFixtureExtension calendarWithName(String name) {
    return new MyCalendarFixtureExtension(name);
  }

  private final String name;

  private MyCalendarFixtureExtension(String name) {
    this.name = name;
  }

  public MyCalendarFixture createFixture(Robot robot, Container root) {
    MyCalendar calendar = robot.finder().findByName(root, name, MyCalendar.class,
true);
    return new MyCalendarFixture(robot, calendar);
  }
}
```

The only method that needs to be implemented is `createFixture(Robot, Container)`, which is the responsible of creating our custom fixture. The static method `calendarWithName(String)` is just a convenience factory method, which we will use to connect our extension to any implementation of `ContainerFixture`:

```
// import static org.fest.swing.sample.MyCalendarFixtureExtension.calendarWithName;

FrameFixture frame = new FrameFixture(new MyFrame());
frame.with(calendarWithName("myCalendar")).selectDate("10/18/08");
```