Quick Start

Quick Start

The following makes it simple to start a griffon project.

- Quick Start
  - Create a Griffon project
  - Create an Application Model
  - Create the Controller Logic
  - Add Content to the View
  - Run the Application
- What's Next

Create a Griffon project

Once you have installed Griffon you can use the built-in target for creating new projects:

```
griffon create-app
```

The target will prompt you for the name of your project and create the project structure below:

```
%PROJECT_HOME%
+ griffon-app
+ conf ---> location of configuration artifacts like builder configuration
+ keys ---> keys for code signing
+ webstart ---> webstart and applet config
+ controllers ---> location of controller classes
+ i18n ---> location of message bundles for i18n
+ lifecycle ---> location of lifecycle scripts
+ models ---> location of model classes
+ resources ---> location of non code resources (images, etc)
+ views ---> location of view classes
+ lib
+ scripts ---> scripts
+ src
+ main ---> optional; location for Groovy and Java source files
(of types other than those in griffon-app/*)
```

Create an Application Model

Make sure you are in the root directory of your project (for argument sake "DemoConsole", a simple script evaluator) by typing

```
cd DemoConsole
```

The "create-app" target created a Griffon MVC Triad for you in the models, views, and controllers directory named after the application. Hence you already have a model class DemoConsoleModel in the models directory.

The application model for the quick start is simple: the script to be evaluated and the results of the evaluation. Make sure to paste the following code into your copy of `DemoConsoleModel`.

DemoConsoleModel.groovy

```
package democonsole

import groovy.beans.Bindable

class DemoConsoleModel {

    String scriptSource
    @Bindable def scriptResult
    @Bindable boolean enabled = true

}
```

Create the Controller Logic

The controller for our quick start app is also simple: throw the contents of the script from the model at a groovy shell. Copy the following code into your controller.

DemoConsoleController.groovy

```
package democonsole

import java.awt.event.ActionEvent

class DemoConsoleController {

    GroovyShell shell = new GroovyShell()

    // these will be injected by Griffon
    def model
    def view

    def executeScript(ActionEvent evt = null) {
        model.enabled = false
        doOutside {
            def result
            try {
                result = shell.evaluate(model.scriptSource)
            } finally {
                edt {
                    model.enabled = true
                    model.scriptResult = result
                }
            }
        }
    }
}
```

The Griffon framework will inject references to the other portions of the MVC triad if fields named `model`, `view`, and `controller` are present in the model or controller. This allows us to access the view widgets and the model data if needed

The executeScript method will be used in the view for the button action. Hence the ActionEvent parameter, and the default value so it can be called without an action event.

Finally, the Griffon framework can be configured to inject portions of the builders it uses. By default, the Threading classes are injected into the controller, allowing the use of the `edt`, `doOutside` and `doLater` methods from the SwingBuilder.

Also, the threading may look a bit obsessive. But good thread management is essential to a well functioning Swing application.

Add Content to the View

The view classes contain the visual components for your application. As with the previous artifacts, copy the contents of the following snippet into its corresponding file: `DemoConsoleView`.

DemoConsoleView.groovy

```
package democonsole

application(title:'DemoConsole', pack:true, locationByPlatform:true) {
    panel(border:emptyBorder(6)) {
        borderLayout()

        scrollPane(constraints:CENTER) {
            textArea(text:bind(target:model, targetProperty:'scriptSource'),
                enabled: bind {model.enabled},
                columns:40, rows:10)
        }

        hbox(constraints:SOUTH) {
            button("Execute", actionPerformed:controller.&executeScript,
                enabled: bind {model.enabled})
            hstrut(5)
            label("Result:")
            hstrut(5)
            label(text:bind {model.scriptResult})
        }
    }
}
```

The view script is a fairly straightforward SwingBuilder script. Griffon will execute these groovy scripts in context of it's UberBuilder (a composite of the SwingBuilder and whatever else is thrown in).

Run the Application

To start your Griffon app run the following target

```
griffon run-app
```

This will run the application as a Java application. You can also use the `run-webstart` target to run the application from a WebStart/JNLP file.

The run-app script implies the execution of the package script. The package script creates file artifacts suitable for a Java application, a WebStart application, and an Applet, with code signed by a self-signed certificate. All from the same source tree. By default they go in the 'staging' directory.

```
ls staging
```

Try out the applet by bringing up the applet.html file in a browser. Or you can try them out directly by running the following commands

```
griffon run-applet
griffon run-webstart
```

What's Next

This is just the first pass at the framework. The sky's the limit! More information about the framework, artifacts, conventions and commands can be found at the Griffon Guide.