Griffon 0.1

Overview

Griffon 0.1 – "Lego Griffon" – is the second major release of Griffon.

Principal Features

Plugin Support

Griffon 0.1 is the first release to port over the Grails Plug in architecture. Being derived from Grails the plugins are very similar in format and layout, but lack some JavaEE tie ins such as configuring from a Servlet Context. Here is a sampler of current Griffon Plugins:

- Development Tools
    - **Code Coverage** - Generates Code Coverage Reports
    - **JDepend** - Dependency Metrics
    - **Scala** - Performs Scala code compilation
- Groovy GUI Builders
    - **Flamingo Builder** - Flamingo Widgets
    - **JIDE Builder** - Jide Widgets
    - **MacWidgets Builder** - MacWidgets widets
    - **SwingX Builder** - SwingX Widgets
    - **SwingXtras Builder** - SwingXtras Widgets (BalloonTip, xswingx, L2fprod)
    - **Tray Builder** - Java6 AWT Tray Widgets
- Runtime Facilities
    - **Installer** - Standalone Installer for Griffon Applications
    - **Splash** - Splash screen
    - **Wizard** - Wizard Support
- Testing
    - **EasyB** - Behavior Driven Development (BDD)
    - **FEST** - GUI Acceptance Testing

MVC Groups Enhancements

The MVC Groups APIs have been enhanced to provide better facilities to encourage good MVC design and to reward it's use.

MVC Groups are created by the `create-mvc <type>`. Files corresponding to the model, view, and controller are created in the directories `griffon-app/model`, `griffon-app/view`, and `griffon-app/controller` respectively. An entry is also added to `griffon-app/conf/Application.groovy` in the mvcGroups section listing the MVCType along with the generated class file names. For example:

samples/FileViewer/griffon-app/conf/Applicaiton.groovy

```
mvcGroups {
    // MVC Group for "FilePanel"
    'FilePanel' {
        model = 'FilePanelModel'
        view = 'FilePanelView'
        controller = 'FilePanelController'
    }
//...
}
```

In this case `FilePanel` is the MVC type name, and the model, view, and controller exist in the default package.

MVC Groups are created in one of two ways, either automatically at startup or explicitly via a call call to `createMVCGroup`. To have a MVCGroup instantiated automatically at startup add the MVC type name in `Applicaiton.groovy` to the list stored in `application.startupGroups`

samples/FileViewer/griffon-app/conf/Application.groovy

```
application {
//...
    startupGroups = ['FileViewer']
//...
}
```

The second way to use MVC Groups in your application is to explicitly create them. In each model, view and controller class created by the framework a meta-method `createMVCGroup` is injected into each instance. The method takes a required MVC type name and an optional name and parameter map as following arguments. Returned is the model, view, and controller instances, for example:

```
int pos = count++
def (model, view, controller)= createMVCGroup('ReusableWidget', "Reuse_$count",
    [parentPanel:view.widgetHolder, pos:count])
```

The name is used to store the reference to the created model, view, controller, and builder in the `app` object (this app object is injected into each model, view, controller and all lifecycle scripts). They are stored in `app.models`, `app.views`, `app.controllers`, and `app.builders` maps respectively. For example, the Greet sample uses the `app` fields to enable the login pane when Greet is done initializing

samples/Greet/griffon-app/lifecycle/Ready.groovy

```
app.controllers.Greet.twitterService = new TwitterService()
app.views.Greet.bind(source:app.controllers.Greet.twitterService,
sourceProperty:'status', target:app.models.Greet, targetProperty:'statusLine')

app.models.LoginPane.loggingIn = false
def focusField = app.views.LoginPane.twitterNameField
if (focusField.text) {
    focusField = app.views.LoginPane.twitterPasswordField
}
focusField.requestFocusInWindow()
```

Finally, there is the `destroyMVCGroup()` method. This method should be called when you are removing a MVC Group from display and will no longer be using it. This method takes the MVC Group name, calls dispose() on the builder, and removes the linkages from the Griffon framework. Of course you still need to insure that there are no links external links to the MVC Group instances to insure that memory remains.

Webstart Enhancements

The Webstart packaging portion of Griffon will now allow you to reference external jars and JNLP files via the `Config.groovy` rather than hard coding them in each individual JNLP file. The lists stored at the values in `griffon.extensions.jarUrls` and `griffon.extensions.jnlpUrls` will be converted to the appropriate JNLP file elements in all of your JNLP files containing the `@jnlpJars@` and `@jnlpExtensions@` variables (the jnlp files Griffon auto-generates already has these, and upgrade will add them as appropriate as well).

For an example of this see James Williams' blog entry Griffon Groks JOGL

Buildtime Events

Buildtime events allow an application or plugin to react to the different phases of the build process, a typical scenario would be to configure the classpath for additional dependencies or perform an additional cleanup routine. The builder plugins use them to copy the required libraries into the build path of the application that installed them for example.

Any griffon script can publish an event, simply call the `event` method with a name and additional parameters as needed. You can also create a special script file that will serve as a holder for all build event handlers, it should be named `_Events.groovy` and it should be placed on the scripts directory of your application. Take the following trivial example

File: scripts/_Events.groovy

```
eventCleanEnd = { -> println "CLEANUP!" }
```

Now every time you invoke the clean command you'll get an additional message when that action has finished its job. Notice the naming convention for the event handler, its name should be of the form `event<EventName>`, including case.

Runtime Events

Similar in nature to build events Griffon applications are now able to publish arbitrary events whenever you need them to. They will also publish events at specific points in their life cycle. Any object that follows the conventions can be registered as an application listener, those conventions are:

- can be a script, a class, a Map or a closure.
- in the case of scripts or classes, they must define an event handler whose name matches on<EventName>, this handler can be a method or a closure property.
- in the case of a map, each key maps to <EventName>, the value must be a closure.
- scripts, classes and maps can be registered/unregistered by calling `addApplicationLister`/`removeApplicationListener` on the app instance.
- closure event handlers must be registered with an overloaded version of `addApplicationLister`/`removeApplicationListener` th at takes <EventName> as the first parameter, and the closure itself as the second parameter.

There is a general, per application, script that can provide event handlers, if you want to take advantage of this feature you must define a script named `Events.groovy` inside griffon-app/conf. Lastly each controller is automatically registered as an application event listener.

Application events can be published in the same way as build events are, just call the `event` method on the application instance like this:

```
class MyController {
  def myAction = { evt = null ->
    // do some processing
    app.event("ProcessingDone",[arg0, arg1])
  }
}
```

Here are some examples of event handlers:

1. Display a message right before default MVC groups are instantiated
File: griffon-app/conf/Events.groovy

```
onBootstrapEnd = { app ->
  println """Application configuration has finished loading.
MVC Groups will be initialized now."""
}
```

2. Print the name of the application plus a short message when the application is about to shut down.
File: griffon-app/controller/MyController.groovy

```
class MyController {
  def onShutdownStart = { app ->
    println "${app.config.application.title} is shutting down"
  }
}
```

3. Print a message every time the event "Foo" is published
File: griffon-app/controller/MyController.groovy

```
class MyController {
  def mvcGroupInit( Map args ) {
    app.addApplicationListener([
      "Foo": {-> println "got foo!" }
    ])
  }

  def fooAction = { evt = null ->
    // do something
    event("Foo",[])
  }
}
```

4. An alternative to the previous example using a closure event handler
File: griffon-app/controller/MyController.groovy

```
class MyController {
  def mvcGroupInit( Map args ) {
    app.addApplicationListener("Foo"){-> println "got foo!" }
  }

  def fooAction = { evt = null ->
    // do something
    event("foo",[])
  }
}
```

Finally, these are the events fired by the application during its life cycle:

- BootstrapEnd
- StartupStart
- StartupEnd
- ReadyStart
- ReadyEnd
- ShutdownStart

Sample Applications

Griffon 0.1 ships with 4 sample applications of varying levels of complexity demonstrating various parts of the framework. In order of complexity they are:

File Viewer

File View is a simple demonstration of creating new MVCGroups on the fly.

WebStart of Application

Source: Subversion Link
To run the sample from source, change into the source directory and run `griffon run-app` from the command prompt.

Font Picker

Font Picker demonstrates form based data binding to adjust the sample rendering of system fonts.
WebStart of Applicaiton

Subversion Link
To run the sample from source, change into the source directory and run `griffon run-app` from the command prompt.

Greet

Greet, a full featured Griffon Application, is a Twitter client. It shows Joint Java/Groovy compilation, richer MVCGroup interactions, and network service based data delivery.
WebStart of Applicaiton

Subversion Link
To run the sample from source, change into the source directory and run `griffon run-webstart` from the command prompt. Because Greet uses JNLP APIs for browser integration using `run-app` will prevent web links from working.

SwingPad

SwingPad, a full featured Griffon Application, is a scripting console for rendering Groovy SwingBuilder views.
An installer will be forthcoming.

Subversion Link
To run the sample from source, change into the source directory and run `griffon run-app` from the command prompt.

0.1.0 Release Notes

Released 9 Mar 2009

Griffon 0.1.0 Resolved Issues (13 issues)

T	Key	Summary
	GRIFFON-16	CreateMvc scripts does not prompt for a name
	GRIFFON-17	TestApp script throws MissingMethodException
	GRIFFON-18	Review plugins before releasing
	GRIFFON-1	Applet Fails Though Swing App Doesn't
	GRIFFON-5	Running <projectname>.launch in Eclipse
	GRIFFON-19	Enable code coverage with FEST/Easyb plugins
	GRIFFON-20	Author should be able to opt out of auto-shutdown
	GRIFFON-21	better error message when passing invalid name to createMVCGroup
	GRIFFON-22	Unknown MVC Types exception erroneously lists the type causing the error
	GRIFFON-37	griffon list-plugin and install-plugin fail
	GRIFFON-8	eclipse reports missing "src/java" and "src/groovy" folder
	GRIFFON-15	run-webstart clears jnlp extensions from jnlp files
	GRIFFON-2	"java.lang.IllegalArgumentException: Compiler configuration must not be null." thrown when a Griffon app is run on Mac

13 issues