Jython Plugin

Description

The Jython plugin enables compiling and running Jython code on your Griffon application.

Installation

The current version of **griffon-jython-plugin** is **0.3**

To install just issue the following command:

```
griffon install-plugin jython
```

Usage

You must place all Jython code under `$appdir/src/jython`, it will be compiled first before any sources available on `griffon-app` or `src/main` which means you can't reference any of those sources from your Jython code, while the Groovy sources can. You will be able to use the Lang Bridge Plugin facilities to communicate with other JVM languages.

Sources placed under `$appdir/src/jython` will generate Java classes available for use within griffon-app.

`$appdir/src/jython/griffon/JythonGreeter.py`

```
from org.codehaus.griffon import IGreeter

class JythonGreeter(IGreeter):
    def __init__(self):
        pass

    def greet(self, who, model):
        greeting = 'Hello %s from Jython!' % str(who)
        print greeting
        model.setOutput(greeting)
```

Where `IGreeter` is an interface allowing Jython to expose methods:

```
package org.codehaus.griffon

public interface IGreeter {
    public void greet(String greetee, MyModel model)
}
```

This will generate a Java class named `griffon.JythonGreeter`, which can be used inside any Griffon artifact, for example a Controller

```
import javax.swing.JOptionPane
import java.beans.PropertyChangeListener
import org.codehaus.griffon.IGreeter
import griffon.jython.JythonObjectFactory

class JyAppController {
    def model
    def view
    def greeter

    def mvcGroupInit(Map args) {
         model.addPropertyChangeListener("output", { evt ->
             if(!evt.newValue) return
             // model.output may have been updated outside EDT
             doLater {
                 JOptionPane.showMessageDialog(app.windowManager.windows[0],
                     evt.newValue, "Yay for Jython", JOptionPane.INFORMATION_MESSAGE)
             }
         } as PropertyChangeListener)

         // Create our JythonGreeter
         JythonObjectFactory factory = new JythonObjectFactory(IGreeter.class,
'JythonGreeter', 'JythonGreeter')
         greeter = (IGreeter) factory.createObject()
     }

    def handleClick = { evt = null ->
        if(!model.input) return
        // clear the result first
        model.output = ""
        // invoke Jython class outside the EDT
        doOutside {
            greeter.greet(model.input, model)
        }
    }
}
```

You are also able to load Jython scripts at any time. By default all scripts placed at `$basedir/griffon-app/resources/jython` will be loaded when the application boostraps itself. For example `griffon-app/resources/jython/fib.py` might look like this:

```
def addNumbers(a, b):
    return a + b
```

With that code in place, the addNumbers function may be executed as a method call on a dynamic property named `py` from a Griffon controller. See below:

```
class FooController {
  def addNumbers = { evt = null ->
     // invoke the function as a method on the 'py' dynamic property
     model.z = py.add_numbers(model.x, model.y)
  }
}
```

The dynamic property will be named `py` by default. The name of the property may be set explicitly in `griffon-app/conf/Config.groovy` by

assigning a value to the griffon.jython.dynamicPropertyName property.

```
griffon.jython.dynamicPropertyName = 'jythonPropertyName'
```

For most applications, the default name of `py` should be fine. You can also alter in which artifacts the property gets injected, by default only controllers will have that property. See `griffon-app/conf/Config.groovy` and look for the following entry

```
griffon.jython.injectInto = ["controller"]
```

Finally, you can load any Jython script by calling `pyLoad(String path)` where `path` will be resolved using Spring's PathMatchingResourcePatternResolver. The default path used during bootstrap is `"classpath*:/jython/*.py"`. It is also worth mentioning that this method will be injected to all artifacts controlled by `griffon.jython.injectInto` and that the prefix `py` will be affected by `griffon.jython.dynamicPropertyName`.

Scripts

- **create-jython-class** - creates a new Jython class in `$basedir/src/jython`.
- **create-jython-script** - creates a new Jython script in `$basedir/griffon-app/resources/py`.
- **jython-repl** - executes a Jython REPL with the application's classpath fully configured.

Source Code

Maintained on GitHub at the griffon-jython plugin repository.

History

Version	Date	Notes
0.3	03-12-12	Griffon 0.9.5+ compatibility
0.2	08-15-11	Griffon 0.9.2 compatibility
0.1	01-04-11	Initial release