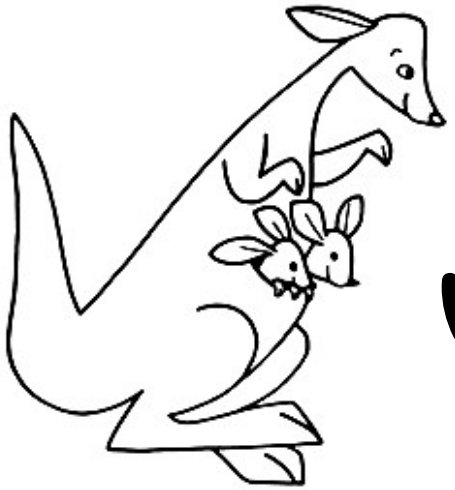
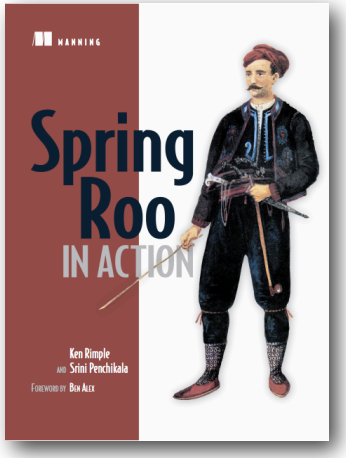
CHARIOT SOLUTIONS

Richmond, Virginia
Java User Group
May 2012



Leaping Ahead with Roo Add-Ons



Ken Rimple
Chariot Solutions

Agenda

- What is Roo?
- Write Roo add-ons
  - Types of add-ons
  - Demo 1: install jQuery
  - Demo 2: install Coffeescript
- Installing and Publishing Add-ons

Recap - What is Spring Roo?

- Command-Line Shell
- Configures Projects
- Provides maven-based builds
- Generates code smartly
- Geared toward Spring projects

Without customization, Roo can configure...

Spring Framework

JPA 2.0

Functional Maven build

Localization

Hibernate, OpenJPA, EclipseLink, or Data Nucleus JPA Providers

NoSQL with MongoDB and Spring Data JPA MongoDB

Persistence JUnit Tests

Tiles layouts and theming

Your pick of RDBMS

Spring JMS w/ActiveMQ

Selenium Web Testing

Reverse Engineering

Email Support

Solr searching

Bean Validators

Spring MVC

Logging w/SLF4J

Spring Services

GWT w/MVP

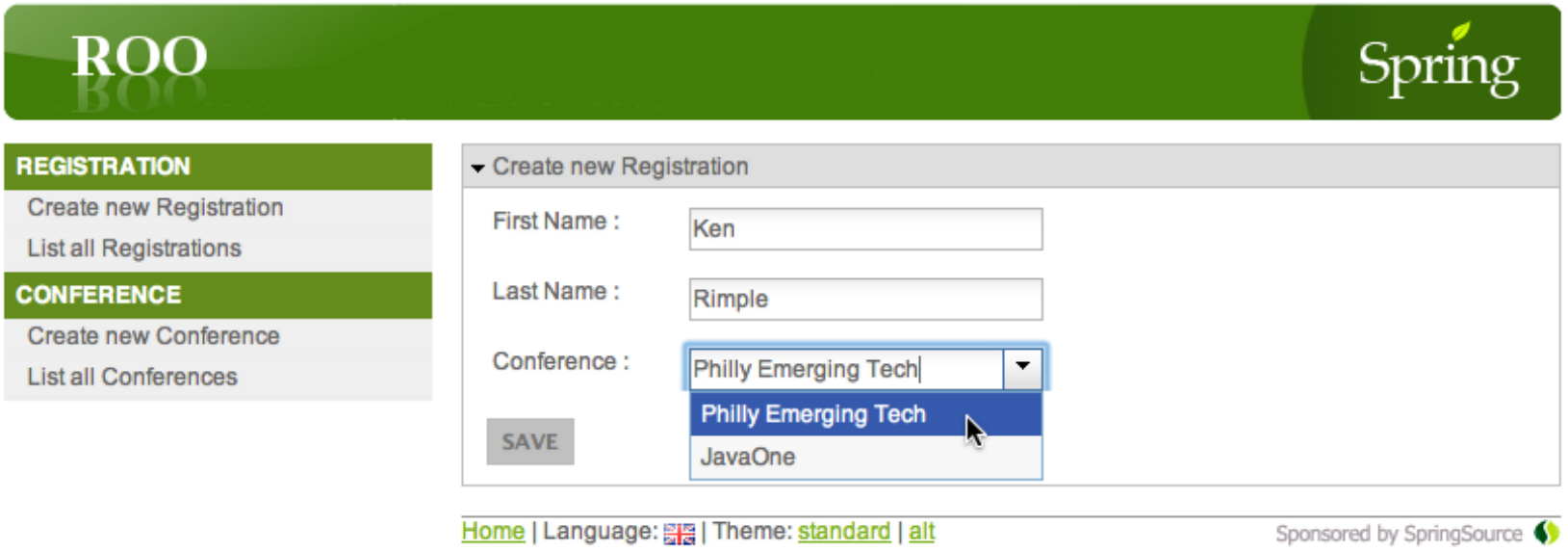
JSF w/Mojarra or MyFaces

Spring Repositories and Spring Data JPA

This script...

```
project --topLevelPackage org.foo --projectName conference-planner
jpa setup --database HYPERSONIC_PERSISTENT --provider HIBERNATE

entity jpa --class ~.model.Conference --testAutomatically
field string --fieldName conferenceName
field date --fieldName startDate --type java.util.Date

entity jpa --class ~.model.Registration --testAutomatically
field string --fieldName firstName
field string --fieldName lastName
field reference --fieldName conference --type ~.model.Conference
      --cardinality MANY_TO_ONE

focus --class ~.model.Conference
field set --fieldName registrations --type ~.model.Registration
      --cardinality ONE_TO_MANY --mappedBy conference

web mvc setup
web mvc all --package ~.web
```

Provides this...

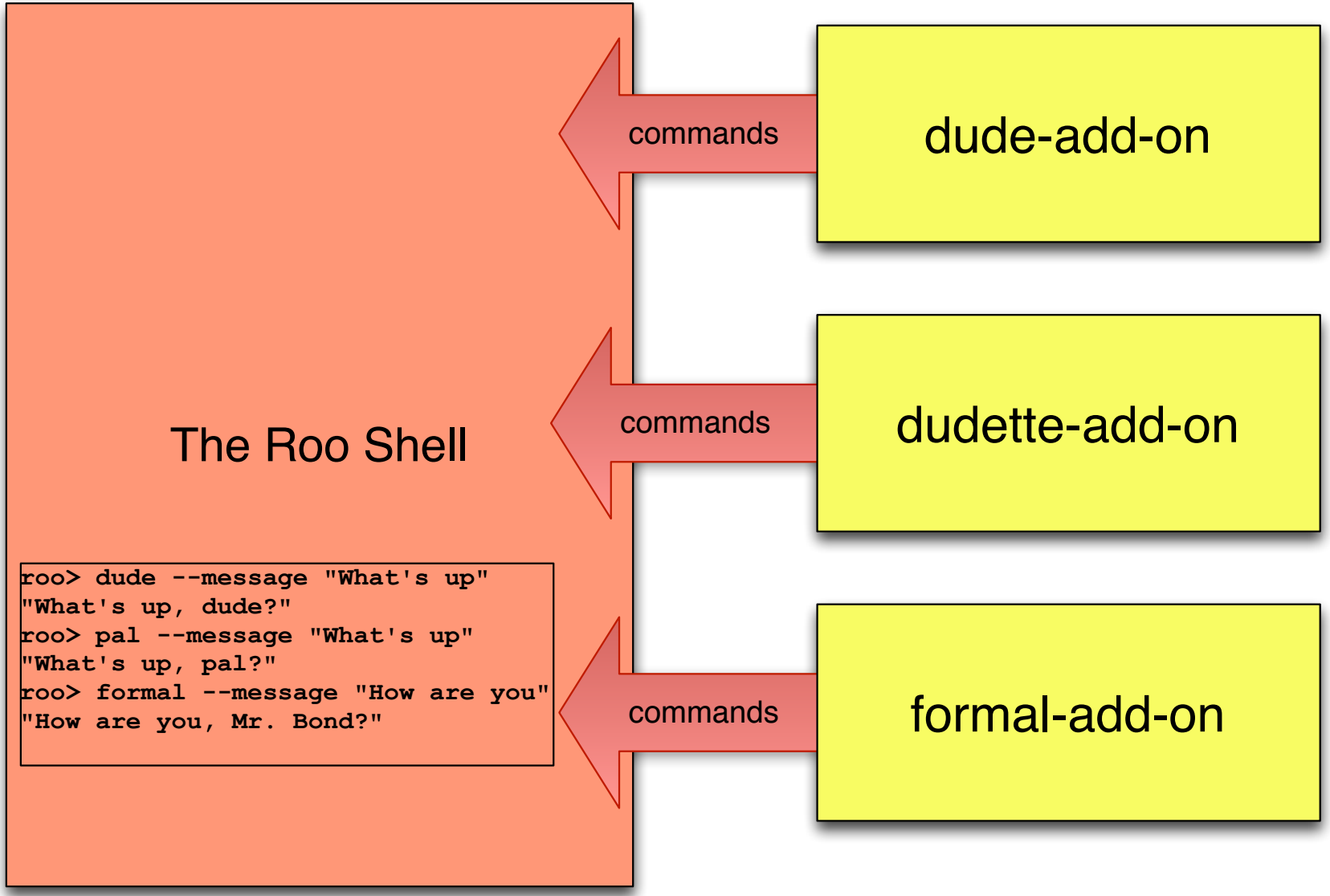


Note - A slight adjustment here, using itemLabel in the create.tagx and update.tagx files to only show conferenceName

What are add-ons?

- OSGi bundle (JAR) projects
- Created in Roo itself
- Built using Maven
- Can be installed in a variety of ways

Roo add-ons inject Shell commands



Why write Add-Ons?

Share Architectural Patterns

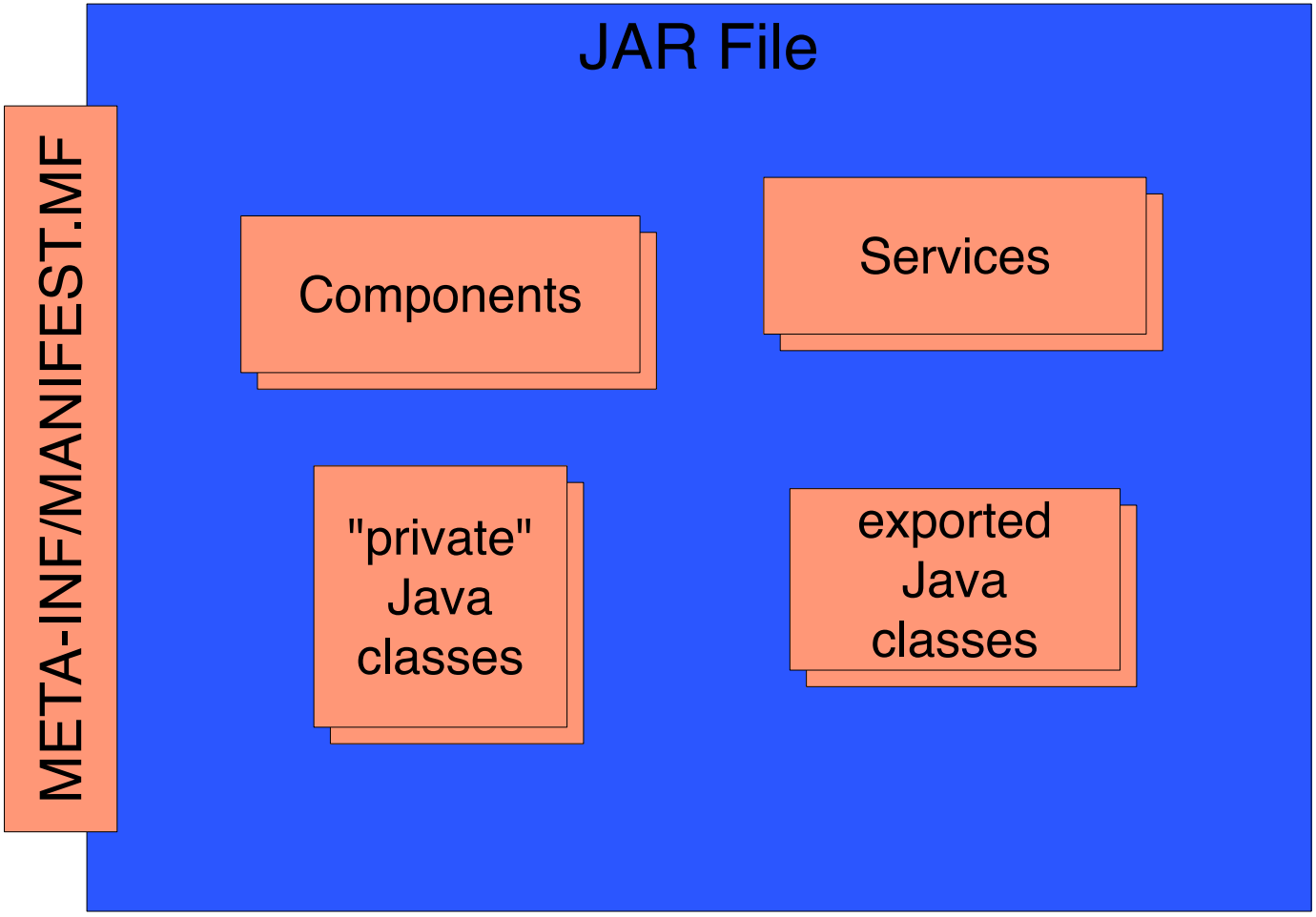
- Your preferred ORM strategy
- Your web platform configuration
- Your set of standard libraries
- An Ajax approach
- Anything you can think of...

Share with the Community

- Configuring your open source projects to work with Roo
- Getting started with your application platform
- Alternative to Maven archetypes

What are Roo add-ons?

Add-ons are OSGi Bundles



Roo is made up of OSGi

- Core platform bundles
- Command/feature bundles
- Roo is an Apache Felix wrapper
  - Runs Felix behind the scenes
  - Exposes and injects OSGi components/services
- This is only for the shell

5 Minutes of OSGi

- OSGi - an open standard for a dynamic module system
- Bundle = Jar with instructions
  - META-INF/MANIFEST.MF
- OSGi Container - platform that manages OSGi bundles - ex: Apache Felix
- OSGi Shell - command line exposed by container

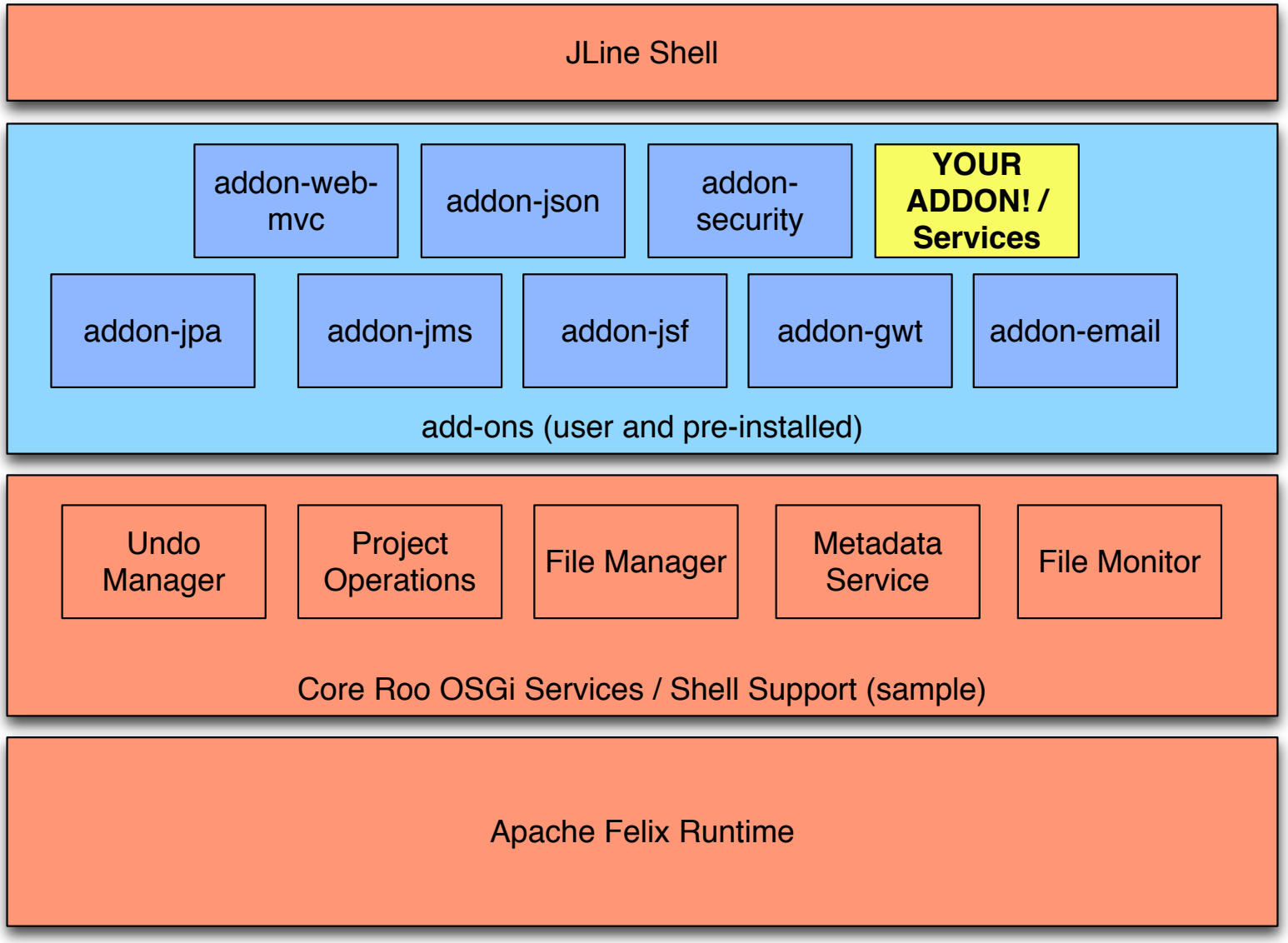
Ok, 6 Minutes...

- OSGi Service - similar to a Singleton Spring Bean
  - But can be loaded/unloaded dynamically
  - Can be deployed automatically when a bundle is loaded
  - Some APIs use annotations like @Component, @Service - similar to Spring (roo uses Felix SCR)

All right, 7...

- But why do I care?
  - Roo Exposes OSGi services (within Bundles) to provide infrastructure
  - Roo add-ons are exposed as OSGi services with a special service
- Bottom line - you need to care a bit about OSGi if you write Roo add-ons

Core Roo Services & Add-ons



CHARIOT SOLUTIONS

Types of add-ons

Type	Use	Examples
simple	Modify files in your project	jQuery add-on web mvc update tags
advanced	Manipulate Maven build	CoffeeScript add-on jms setup, etc...
i18n	Add a new scaffold language	Installing the Norwegian language
wrapper	Jar contents added to the Roo shell classpath	Installing JDBC drivers for DBRE

Simple add-ons – jQuery

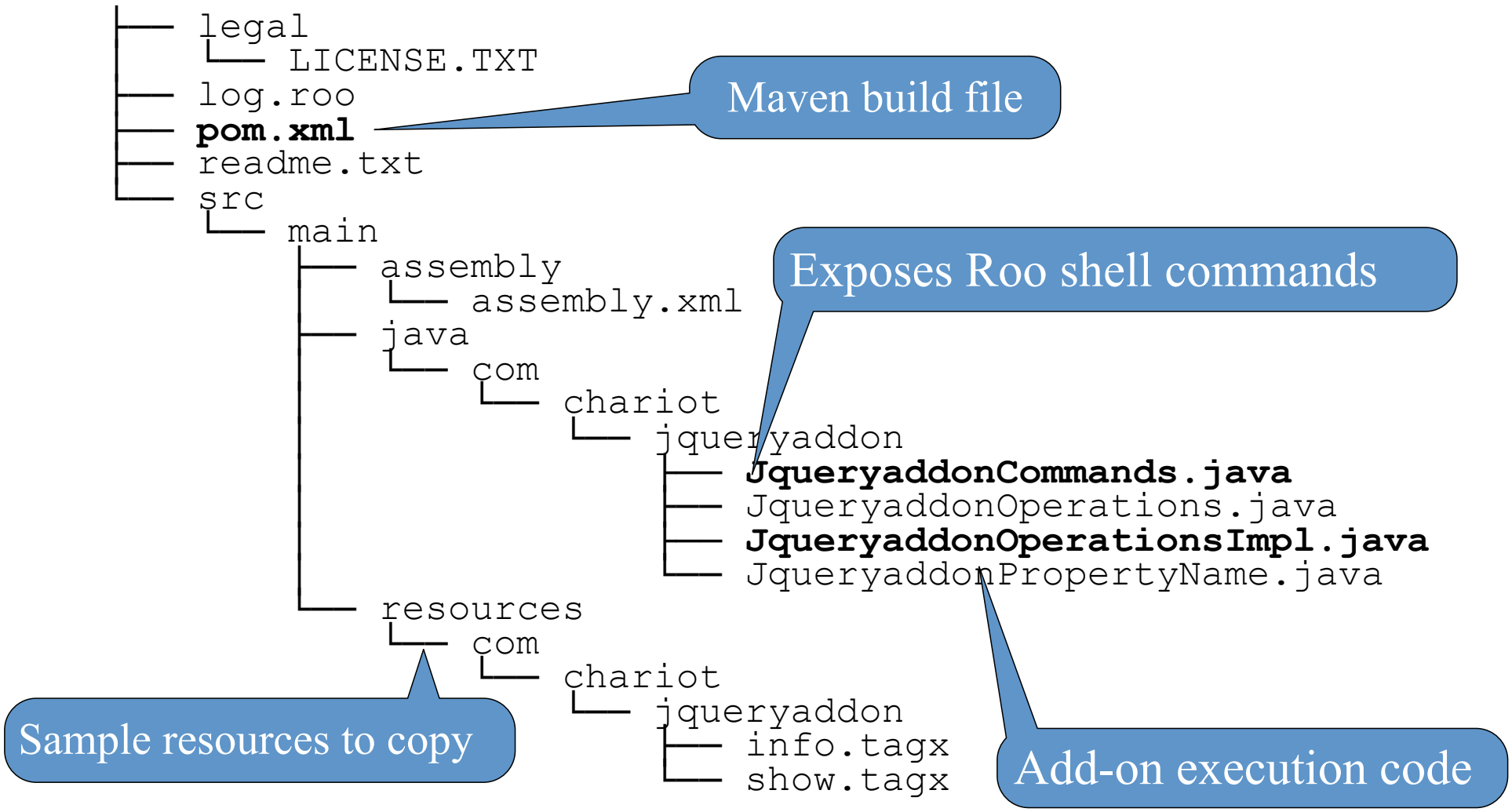
CHARIOT SOLUTIONS

Creating the add-on

- Create empty directory
- Fire up Roo
- Issue add-on create command (single line):

```
roo> addon simple create
   --projectName myProject
   --topLevelPackage com.chariot.jqueryaddon
```

Simple Addon Files



Let's Get Started

- Scenario #1
  - Install a stub jQuery command
  - Echo "jQuery Installed" when **jquery setup** is issued
  - This gives us basic Hello-world functionality

Command Object

- Gut and replace with:

```
@Component
@Service
public class JqueryaddonCommands implements CommandMarker {

    private Logger log = Logger.getLogger(getClass().getName());

    @CliCommand(value = "jquery setup", help = "Setup jQuery")
    public void sayHello() {
        log.severe("jQuery installed.");
    }
}
```

Let's Get Injected

- Let's use that Operations class
  - Replace the interface with:

```
package com.chariot.jqueryaddon;

public interface JqueryaddonOperations {
    void setup();
}
```

The Implementation

- Steps
  1. Put a copy of jQuery in our **src/main/resources** directory structure
  2. Inject our Operations Service into our Commands Service Component
  3. Call the Operations Service's **setup()** method from the command method
  4. Copy the files in the setup method

Step 1 - Load up jQuery

- Remove those silly tag files & replace with a jQuery file:

```
src/main/resources
└── com
    └── chariot
        └── jqueryaddon
            └── jquery-1.7.2.min.js
```

Inject and Call our Operations Component/Service

- Add injection to your Command:

```
@Component
@Service
public class JqueryaddonCommands implements CommandMarker {

    private Logger log = Logger.getLogger(getClass().getName());

    @Reference private JqueryaddonOperations addonOperations;

    @CliCommand(value = "jquery setup", help = "Setup jQuery")
    public void setup() {
        addonOperations.setup();
    }
}
```

Step 3 - Copy the file

- Replace the methods with (keep the various @Reference variables and Path separator at the top):

```
public void setup() {
   PathResolver pathResolver =
      projectOperations.getPathResolver();

   String jsDirectory = pathResolver.getFocusedIdentifier(
        Path.SRC_MAIN_WEBAPP, SEPARATOR + "javascript");

   if (!fileManager.exists(jsDirectory)) {
      fileManager.createDirectory(jsDirectory);
   }
   createOrReplaceFile(jsDirectory, "jquery-1.7.2.min.js");
}
```

createOrReplaceFile....

```
private void createOrReplaceFile(String path, String fileName) {

  String targetFile = path + SEPARATOR + fileName;

  InputStream inputStream =
      FileUtils.getInputStream(getClass(), fileName);

  MutableFile mutableFile = fileManager.exists(targetFile) ?
    fileManager.updateFile(targetFile) :
    fileManager.createFile(targetFile);
    OutputStream targetFileStream = mutableFile.getOutputStream();
    try {
      IOUtils.copy(inputStream, targetFileStream);
    } catch (IOException e) {
      throw new IllegalStateException(e);
    } finally {
      IOUtils.closeQuietly(inputStream);
      IOUtils.closeQuietly(targetFileStream);
    }
  }
```

Step 4 - Install it

- First, build the add-on project:

```
$ mvn clean package
```

- Then, install it (uninstalling first if you're testing):

```
# Optionally, uninstall:
roo> osgi uninstall --bundleSymbolicName com.chariot.jqueryaddon

# install... (one line)
roo> osgi start --url file:///Users/kenrimple/foo/target/
              com.chariot.jqueryaddon-0.1.0.BUILD-SNAPSHOT.jar
```

Now, set it up and run it!

- From another project do this:

```
roo> jquery setup
Updated SRC_MAIN_WEBAPP/javascript/jquery-1.7.2.min.js
```

- Need a scratch project?

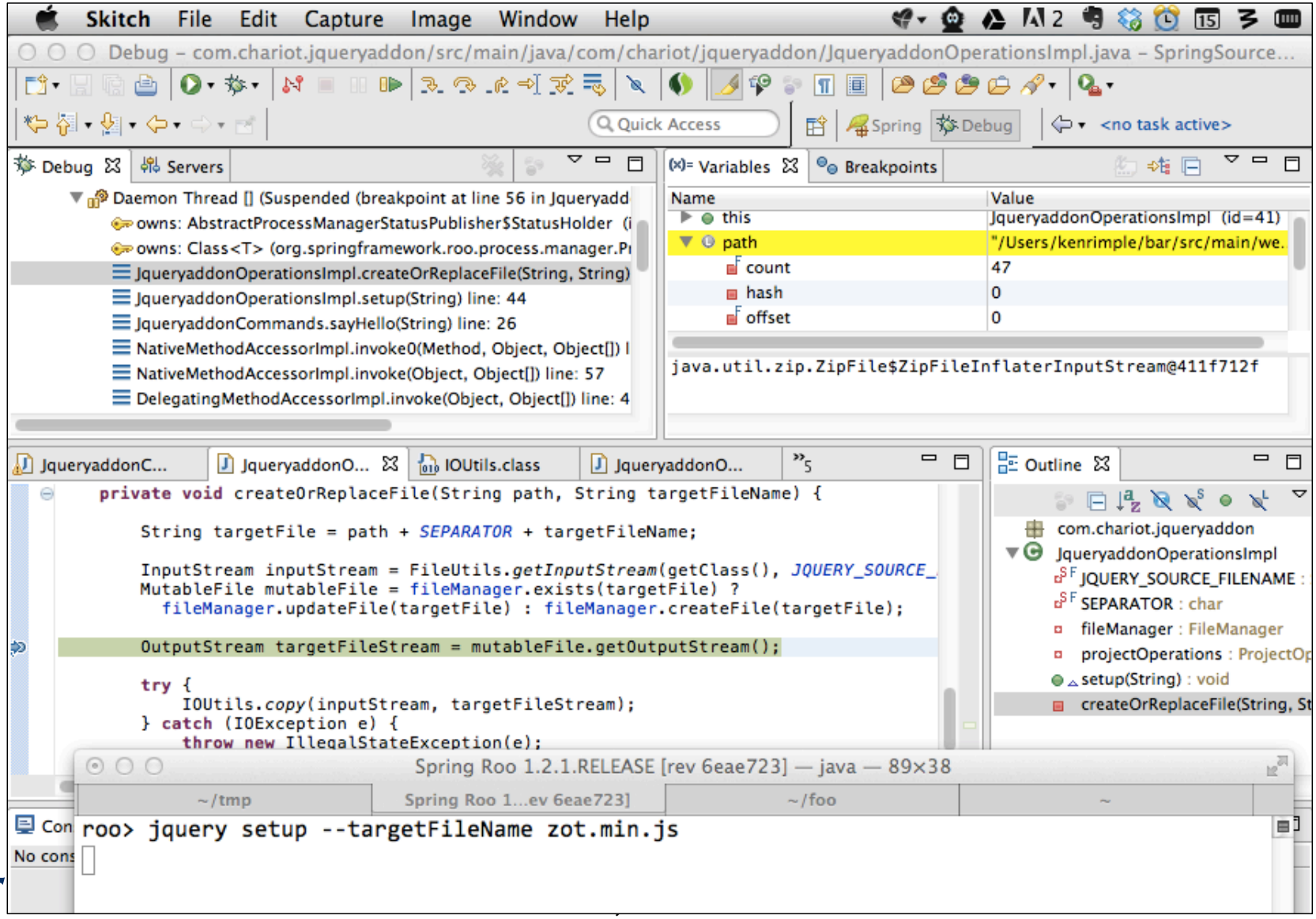
```
$ roo script wedding.roo
```

Debugging Roo Add-ons

- Easy. Just edit your Roo script and add standard JDWP debugging to the java command at the bottom!

```
...
java -Xdebug \
-Xrunjdwp:transport=dt_socket,address=8000,server=y,suspend=n \
-Dis.apple.terminal=$APPLE_TERMINAL...
...
```

Attach w/IDE and add-on project



CHARIOT SOLUTIONS

Speaking of Parameters...

- Here's how to add Parameters to your Roo scripts
  - Step 1 - Add @CliOption for each parameter in the Command
  - Step 2 - Pass the variable to your operations method

Adding a @CliOption - target file name

- Specify parameter settings before each method parameter

```
@CliCommand(value = "jquery setup",
    help = "Setup jQuery")
public void setup(
  @CliOption(key = "targetFileName",
    mandatory = false,
    specifiedDefaultValue = "jquery-1.7.2.min.js",
    unspecifiedDefaultValue = "jquery-1.7.2.min.js",
    help = "Name of target javascript file")
    String targetFileName) {
      addonOperations.setup(targetFileName);
}
```

Implementation...

```
public void setup(String targetFileName) {
  PathResolver pathResolver =
     projectOperations.getPathResolver();

  String jsDirectory = pathResolver.getFocusedIdentifier(
     Path.SRC_MAIN_WEBAPP, SEPARATOR + "javascript");

  if (!fileManager.exists(jsDirectory)) {
    fileManager.createDirectory(jsDirectory);
  }

  createOrReplaceFile(jsDirectory, targetFileName);
}
```

Implementation...

```
private void createOrReplaceFile(
             String path, String targetFileName) {

  String targetFile = path + SEPARATOR + targetFileName;

  InputStream inputStream = FileUtils.getInputStream(
    getClass(), JQUERY_SOURCE_FILENAME);

  MutableFile mutableFile = fileManager.exists(targetFile) ?
    fileManager.updateFile(targetFile) :
    fileManager.createFile(targetFile);

  OutputStream targetFileStream =
     mutableFile.getOutputStream();
...
```

Questions so far?

- We saw how to
  - Create a simple add-on
  - Copy files into the project
  - Expose commands
  - Add parameters
  - Install Add-ons with OSGi
  - Debug

Let's take a mental break...

Geek Quiz!

Question #1

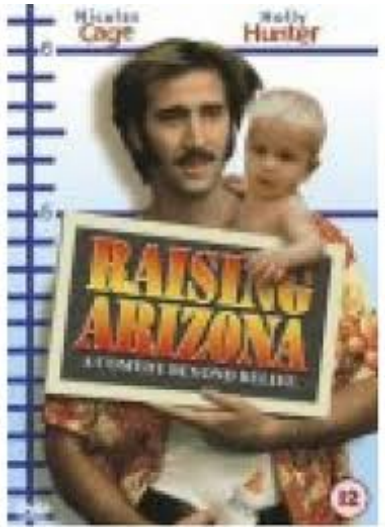
- How many instances can there be of a singleton bean in a Spring Container?
  - A) One per class definition
  - B) One per bean definition
- Answer:
  - B - Because each Bean is configured individually, you can mount it multiple times in XML

Question #2

- How many instances are created of a Roo OSGi Component / Service?
  - A) One per bundle
  - B) One per container
  - C) Acts as a prototype
- Answer:
  - B - because Roo implements them as simple Singletons

Question #3

- **Who was the better Spock?**
- A) Leonard Nimoy
- B) Zachary Quinto
- C) Dr. Spock?
- According to Raising Arizona, C



Question #4 - Javascript!

- What does this statement return:

```
val = 23;
var x = function(val) { return val * 12; }
console.log("val * 12 is ", x.call(this, 12));
console.log("val is", val);
```

- Answer: "val * 12 is 144" and "val is 23"
- Why?

Enough fun... Back to work!

A few more basics

- How about a shortcut for updating an OSGi add-on?
- You don't have to uninstall / install, just do osgi update:

```
# (one line)
roo> osgi update
   --url file:///Users/kenrimple/foo/target/
      com.chariot.jqueryaddon-0.1.0.BUILD-SNAPSHOT.jar
```

Resetting OSGi bundles

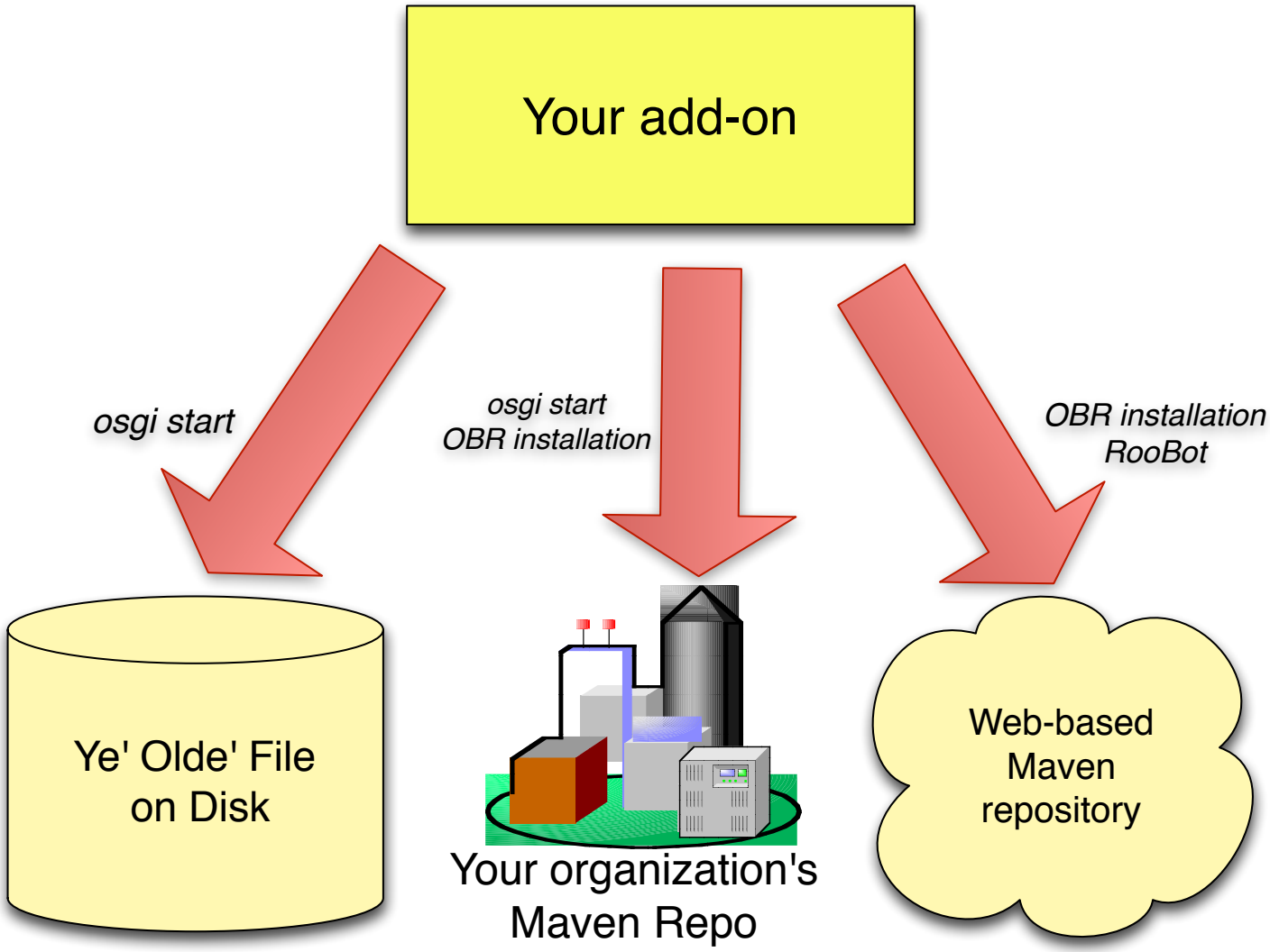
- If your Roo installation acts badly, and add-ons are at fault, do this:
  - Delete <ROO_INSTALLATION>/cache
- This
  - Uninstalls all add-ons
  - Clears any bundles loaded from the cache

Learn by Reading

- You can read up on many of the add-ons used by Roo just by reviewing the source code

Deployment Options

Add-on Deployments



Local Deployment

We've already done this

- Just use the Roo osgi start command:

```
roo> osgi start --url
     file:///.../addonbundle.jar
```

  - But each developer has to do this themselves and you must d/l the files

Deploying using an OBR

CHARIOT SOLUTIONS

What is an OBR?

- An XML-based OSGI Bundle Repository
  - e.g. - Maven Repository with XML descriptor linking to bundles
  - Maintained by Roo deployments

The OBR command set

```
# add an OSGi OBR repository
osgi obr url add --url url

# load an add-on from a repository
osgi obr start --bundleSymbolicName bsn

# find an OBR add-on from
# the set of installed repos
osgi obr list --keywords "coffeescript"

# refresh your OBR
osgi obr refresh --url url

# remove an OBR repository
osgi obr remove --url url
```

How to deploy to an OBR?

CHARIOT SOLUTIONS

```
<!-- We are using CloudBees' GIT -->
<scm>
  <connection>
   scm:git:ssh://git@git.cloudbees.com/...
     sillyweasel/jquery-roo-addon.git
  </connection>
  <url>
   git:ssh://git@git.cloudbees.com/...
     sillyweasel/jquery-roo-addon.git
  </url>

  <developerConnection>
  scm:git:ssh://git@git.cloudbees.com/...
     sillyweasel/jquery-roo-addon.git
  </developerConnection>
</scm>
```

Setting up OBR Deploy

```
...
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  ...
  <configuration>
    <instructions>
      <Bundle-SymbolicName>
         ${project.artifactId}
      </Bundle-SymbolicName>
      <Bundle-Copyright>
         Copyright ${project.organization.name}.
         All Rights Reserved.
      </Bundle-Copyright>
      <Bundle-DocURL>${project.url}</Bundle-DocURL>
      <Export-Package/>
    </instructions>
  ...
```

The OBR Bundle Location

```
   ...
    <remoteOBR>true</remoteOBR>
       <bundleUrl>
        httppgp://repository-sillyweasel.forge.cloudbees.com/
          release/org/sillyweasel/addons/coffeescript/
          org.sillyweasel.addons.coffeescript/
          ${project.version}/
          ${project.artifactId}-${project.version}.jar
      </bundleUrl>
  </configuration>
</plugin>
```

Consuming from an OBR

Mount an OBR

- You can mount an OBR using
  - **osgi obr url add --url** url
- Then you can use
  - **osgi obr list** to see the list of addons

```
roo> osgi obr url add --url
   http://repository-sillyweasel.forge.cloudbees.com/release/
   repository.xml
roo> osgi obr list
coffeescript-addon [org.sillyweasel.addons.coffeescript]
  (0.9.12, 0.9.9)
jqueryui [org.sillyweasel.addons.jqueryui]
  (0.9.9, 0.9.5, 0.9.4, 0.9.3, 0.9.1, 0.9.0.RELEASE)
org.sillyweasel.addons.site [org.sillyweasel.addons.site]
  (0.1.0)
```

Installing an Add-On

- From an OBR, use **osgi obr start**:
  - Optionally you can use a version #, otherwise you get the latest one.

```
# all one line...
roo> osgi obr start --bundleSymbolicName
        org.sillyweasel.addons.jqueryui;0.9.9
Target resource(s):
-------------------
   jqueryui (0.9.9)

Deploying...done.
```

The RooBot

- For adding to the Roo public repository
- Email the URL to your OBR to s2-roobot@vmware.com
- Roo team will build your add-on, deploy it to OBR
- Check for build errors at http://spring-roo-repository.springsource.org/roobot/roobot-log.txt

Advanced Roo add-ons

Advanced Add-Ons

- Can configure the Maven POM
- Can Create, Remove ITDs
- Can respond to file create, update, removal events
- Can modify existing Java class files

Actually, so can Simple ones

- These are just templates
  - You can add some of the advanced features to an add-on you started as a simple one...
  - Just mount the appropriate Roo service and call it
- This just gets you started

Do you dislike Javascript?

- Coffeescript is an alternative
  - Simplifies Javascript
  - Ruby-ish syntax
  - Removes a lot of strange things from Javascript
    - Always uses === (not ==)
    - Always scopes properly
    - Generates "safe" javascript

Let's create an add-on

- The CoffeeScript add-on
  - Drop Coffeescript files in
    - /src/main/webapp/scripts
  - Run your build, it drops 'em in
    - /src/main/webapp/scripts
    - as Javascript files
  - We'll use the Coffee Maven Plugin from Talios

From the Talios GitHub page

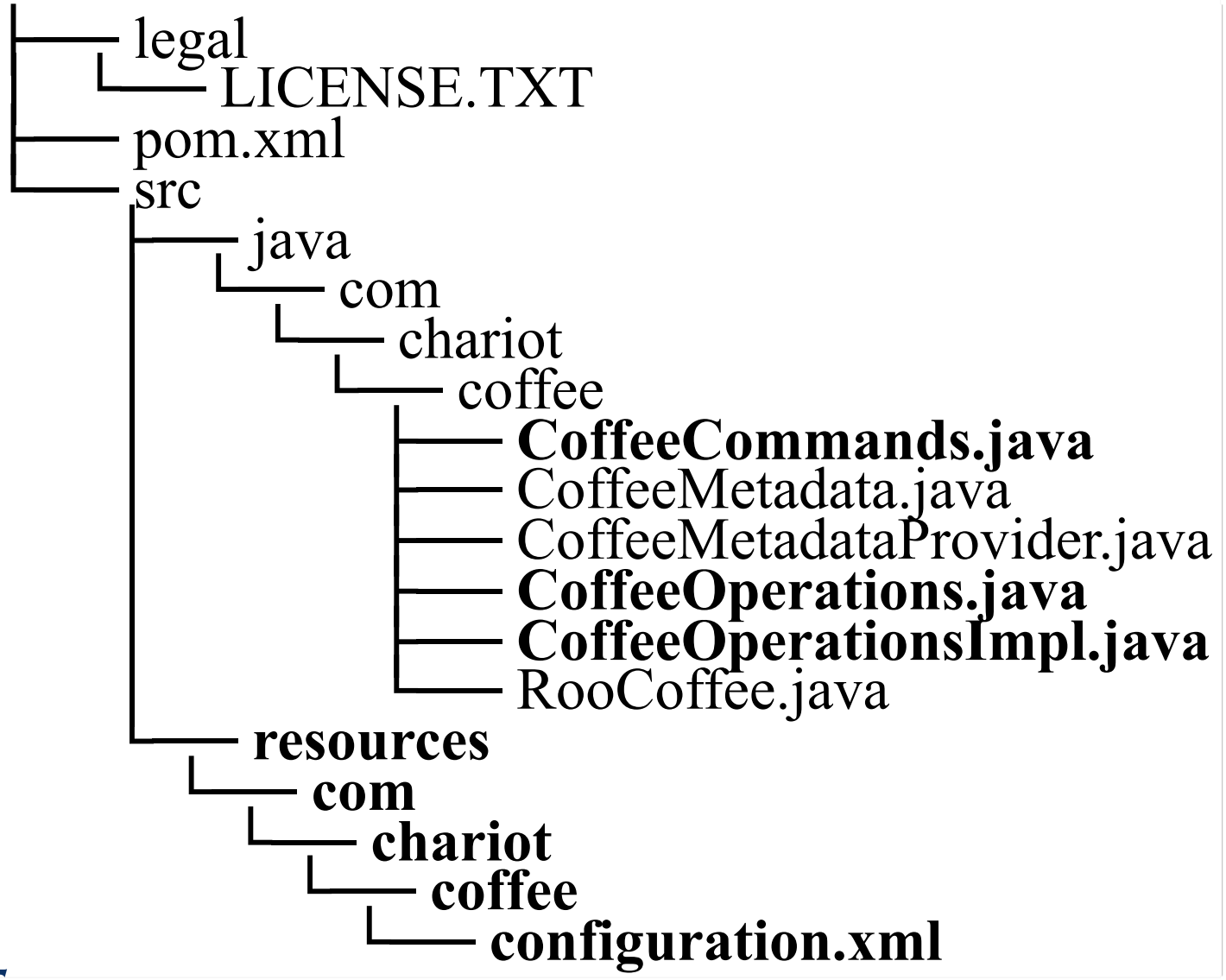
- https://github.com/talios/coffee-maven-plugin
- Steps
  - Mount plugin
  - Configure Executions (attach to build)
  - Configure and customize paths

First, we create the add-on

- Just use the Roo **addon advanced create** command:

```
roo> addon advanced create
   --projectName coffee-addon
   --topLevelPackage com.chariot.coffee
```

The Add-on Files



So, mostly like before

- Except that now we have a **configuration.xml** file in there
  - It's used for storing XML configuration
  - Maven, Spring, other systems use XML
- To fetch, use this:

```
XmlUtils.getConfiguration(getClass())
```

Project Service!

- To manipulate our project, we need to call a Roo service
- The Roo ProjectManager
  - adds, removes, updates Maven dependencies, properties, plugins, and other items
- We'll use it to install our Maven plugin...

Set up our Commands

```
@Component
@Service
public class CoffeeCommands implements CommandMarker {

  @Reference
  private CoffeeOperations operations;

  @CliAvailabilityIndicator("coffee setup")
  public boolean isCommandAvailable() {
    return operations.isCommandAvailable();
  }

  @CliCommand(value = "coffee setup", help = "Setup Coffee addon")
  public void setup() {
    operations.setup();
  }
}
```

The Operations Interface

- Two interface methods...
- Approached just like Spring...

```
public interface CoffeeOperations {
   boolean isCommandAvailable();
   void setup();
}
```

Implementation - Detection

```
@Component
@Service
public class CoffeeOperationsImpl implements CoffeeOperations {

  @Reference
  private ProjectOperations projectOperations;

  public boolean isCommandAvailable() {

    if (!projectOperations.isFocusedProjectAvailable()) {
      return false;
    }
    Plugin coffeePlugin = new Plugin("com.theoryinpractise",
        "coffee-maven-plugin", "1.0.0");

    return projectOperations.getFocusedModule()
        .getBuildPluginsExcludingVersion(coffeePlugin).size() == 0;
  }
...
```

Our Configuration.xml file

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration>
  <plugin>
    <groupId>com.theoryinpractise</groupId>
    <artifactId>coffee-maven-plugin</artifactId>
    <version>1.4.0</version>
    <executions>
      <execution>
        <id>coffee</id>
        <phase>compile</phase>
        <goals>
          <goal>coffee</goal>
        </goals>
      </execution>
    </executions>
    <configuration>
      <coffeeOutputDirectory>src/main/webapp/javascript/</coffeeOutputDirectory>
      <coffeeDir>src/main/webapp/javascript/</coffeeDir>
       <bare>false</bare>
   </configuration>
  </plugin>
</configuration>
```

Implementation - setup()

```
public void setup() {
  Element configuration = XmlUtils.getConfiguration(getClass());
  Plugin coffeePlugin = new Plugin(
      XmlUtils.findFirstElement("//plugin", configuration));

  projectOperations.addBuildPlugin(
      projectOperations.getFocusedModuleName(), coffeePlugin);
}
```

Demonstrate Coffee Plugin

Testing and add-ons

Testing Options

- Unit testing - standard JUnit tests
  - Mockito works fine for mocking
  - A few slight adjustments for mocking

Adjustments

- Since Roo injects services into private objects with no setters/ getters
  - Widen these to package visibility so unit tests can mock them
- You need to add JUnit and your favorite mocking library to add-ons

Upcoming Features

- In Roo 1.3
  - Bug fixes
  - The tailor add-on
- Six week minor release cycles in the future
  - Roo 1.4 should be out by end of summer
- Plenty of cycles to file enhancement JIRAs and get them addressed!!!

Add-on "Best Practices"

- If you provide a setup, you should provide a remove method to un-do if possible
- If you write files, ALWAYS use the File Manager to create a Mutable File
- Always close your files so they write
- Use Validate to assert safety of plugin operations

Add-on "Best Practices"

- If you run into a problem in an add-on, throw an InvalidStateException
  - Don't take a long time to respond to @CliAvailabilityIndicator requests (slows down the shell)
  - Don't expect network connectivity and handle it when it isn't there

Add-on "Best Practices"

- Don't make your add-ons so course-grained that they can't be used compositionally
- Choose some sensible defaults
- Provide help for your parameters
- Provide default typed-in values where possible

Some additional uses of addons

- Replacement for Maven's Archetypes
  - Can be applied to existing projects
  - Can be added/removed
  - Can be used piecemeal
- Your team's APIs or platform stack samples

Other add-on features

- Can annotate classes (see the Roo JSON add-on)
- Can respond to classfile changes and generate / remove / update ITDs
- Can parse / modify XML and JSPX files using JAXP

Other add-on features

- Can trigger maven or operating system commands
- Can add XML files, Spring configuration files, and add sections to configuration
- Anything you can think up!

The Roo community needs

- More web framework addons
- More JMS container add-ons
- More NoSQL DBMS support
- Better client-side widgets
- Better examples to start with
- More useful add-on services (JIRA)

Summary

- Start with a simple add-on (using roo addon create simple)
  - Good examples: addon-backup, addon-logging
- Learn how to deploy add-ons
  - Roo in Action chapter 12
  - This presentation
- Experiment with advanced add-ons (ones that manipulate the project configuration or Aspects
    - addon-jms, addon-email, addon-equals

Read the Source Code!!

- https://github.org/springsource/spring-roo
- All Roo features are Roo add-ons

Get involved!

- Build an add-on
- Host it on github and ask developers to use an OBR with **osgi start**
- Test existing add-ons, report broken ones to the Roo team
- If adventurous, use public Nexus, Google Code, CloudBees, other foundries and host an OBR

Q&A & Resources

- Of course, Roo in Action (chapters 11 & 12)
- Slides available at : chariotsolutions.com/presentations
- Manning Roo in Action forum (manning.com/springrooinaction)
- Springsource Roo forum at forum.springsource.org
- @RooInAction, @krimple, @SpringRoo, @WeaselForge
- Chariot blogs (blog.chariotsolutions.com) and my Roo bloggings at rimple.com
- Silly Weasel Roo add-ons @ rimple.com/silly-weasel
- Spring Roo podcast (coming soon)