Springing Forward with Roo Add-ons

- Ken Rimple, co-author, Spring Roo in Action
- Twitter: @krimple, Blog: http://rimple.com

About me - Ken Rimple (@krimple)

- Co-Author, Spring Roo in Action w/Srini Penchikala
- Host Silly Weasel Roo add-ons project (rimple.com/silly-weasel-forge)
- Director of Education Services, Chariot Solutions
  - Certified instructor, mentor, consultant
  - I run chariotsolutions.com/education
  - Podcasts, screencasts, etc., at emergingtech.chariotsolutions.com
  - Chariot is a VMWare Authorized Training Center
  - We teach Spring, Maven, other courseware
  - We also develop in Spring, Java, mobile, integration technologies
- More about me at my blog rimple.com/about-me

What is Spring Roo?

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

Set up a Spring Project

```
roo> project --topLevelPackage org.foo
   --projectName conference-planner

roo> jpa setup --database HYPERSONIC_PERSISTENT
    --provider HIBERNATE
```

Set up JPA Entities

```
roo> entity jpa --class ~.model.Conference
    --testAutomatically

roo> entity jpa --class ~.model.Registration
    --testAutomatically
```

Set up Conference Fields

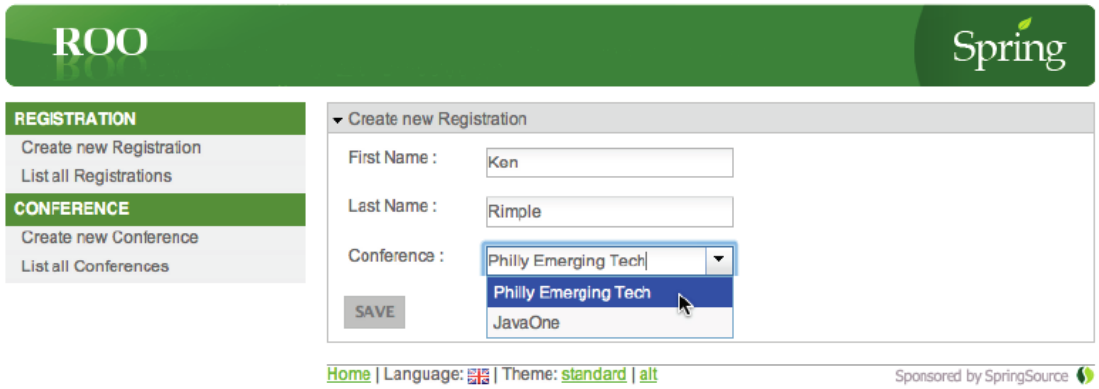
```
roo> focus --class ~.model.Conference

roo> field string --fieldName conferenceName

roo> field date --fieldName startDate
     --type java.util.Date

roo> field set --fieldName registrations
     --type ~.model.Registration
     --cardinality ONE_TO_MANY --mappedBy conference
```

Set up Registration Fields

```
roo> focus --class ~.model.Registration

roo> field string --fieldName firstName

roo> field string --fieldName lastName

roo> field reference --fieldName conference
   --type ~.model.Conference
   --cardinality MANY_TO_ONE
```

Set up Web MVC

```
roo> web mvc setup

roo> web mvc all --package ~.web
```


ROO
Spring
REGISTRATION
Create new Registration
List all Registrations
CONFERENCE
Create new Conference
List all Conferences
Create new Registration
First Name :
Ken
Last Name :
Rimple
Conference :
Philly Emerging Tech
Philly Emerging Tech
JavaOne
SAVE
Home | Language: | Theme: standard | alt
Sponsored by SpringSource

Roo can configure...

Functional Maven build * Spring Framework * JPA 2.0 * Bean Validators * Hibernate, OpenJPA, EclipseLink or Data Nucleus * JPA Providers * Spring Services * Spring Repositories and Spring Data JPA * NoSQL with MongoDB and Spring Data JPA MongoDB * Spring JMS w/ActiveMQ * Email Support * Spring MVC * Solr searching * JSF w/Mojarra * GWT w/MVP or MyFaces * Tiles layouts and theming * Localization * Your pick of RDBMS * Reverse Engineering * Functional Maven build * Logging w/SLF4J * Selenium Web Testing * Persistence JUnit Tests...

These are installed via Roo Add-Ons

- Java-based recipes
- Some installed with Roo, others are downloadable
- You can go further... and write your own add-ons!

What are Roo Add-Ons?

- Components that can add commands and behavior to a running Roo Shell
  - OSGi bundle (JAR) projects
  - Created in Roo itself
  - Built using Maven

Can be installed in a variety of ways

- Locally via JAR
  - `roo> osgi start --url` *`path`*
- Object Bundle Repository
  - `osgi obr` command set
- Roo public add-on registry
  - `addon` command set

Add-on Project Types

- Four types of add-ons available
  - Simple
  - Advanced
  - i18n
  - wrapper
- We'll focus on the simple and advanced ones

Simple add-ons

```
roo> addon create simple --projectName simpleaddon
     --topLevelPackage org.foo.bar
```

- Modifies files in your project structure
- Example add-on: jQuery add-on
- Features used: modifies mvc tags and adds jQuery library

Advanced add-ons

```
roo> addon create advanced --projectName advancedaddon
     --topLevelPakage org.foo.bar
```

- Adds maven manipulation, base class for re-use of framework
- Example: CoffeeScript add-on
- Features used: transactional file copy, maven build configuration changes

i18n add-ons

- Creates a new language bundle
- The language bundle can be installed
- Adds support for additional languages
- Example: support the Norwegian language

wrapper add-ons

- Builds an OSGi-compliant JAR
- Wraps an existing Maven JAR's contents
- Allows the JAR to be loaded into the Roo shell at runtime
- This may be required for JDBC drivers, other features needed by shell add-ons
- Example: Wrapper around JDBC Driver for DBRE

Add-on API key features

- Injecting Shell commands
- Providing Events to other Shell add-ons
- Subscribe to and publish to the Meta-Data Service
- Write, edit, and manipulate ITDs and Java classes

What are ITDs?

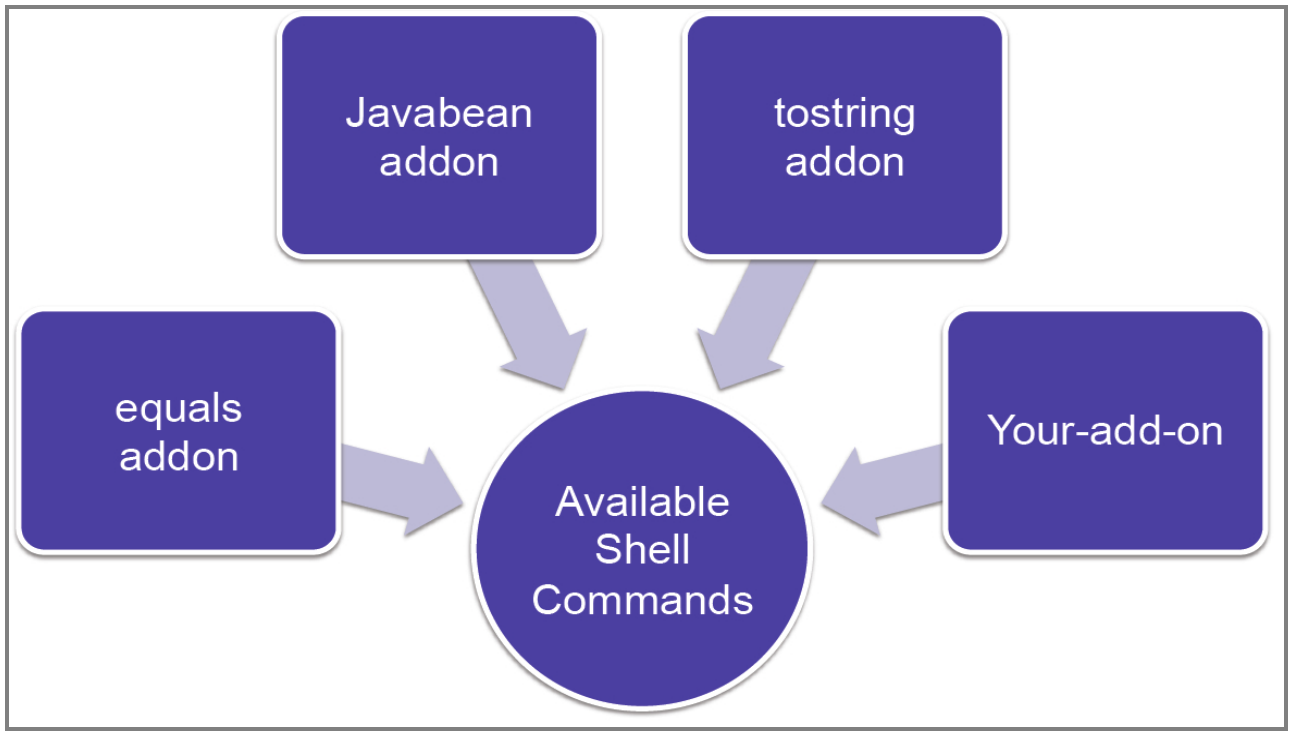
- Inter-Type Declarations
  - AspectJ-generated class fragments
  - Woven into existing classes
- Separates features from boilerplate
- 100% compile-time - no runtime component

Sample ITD

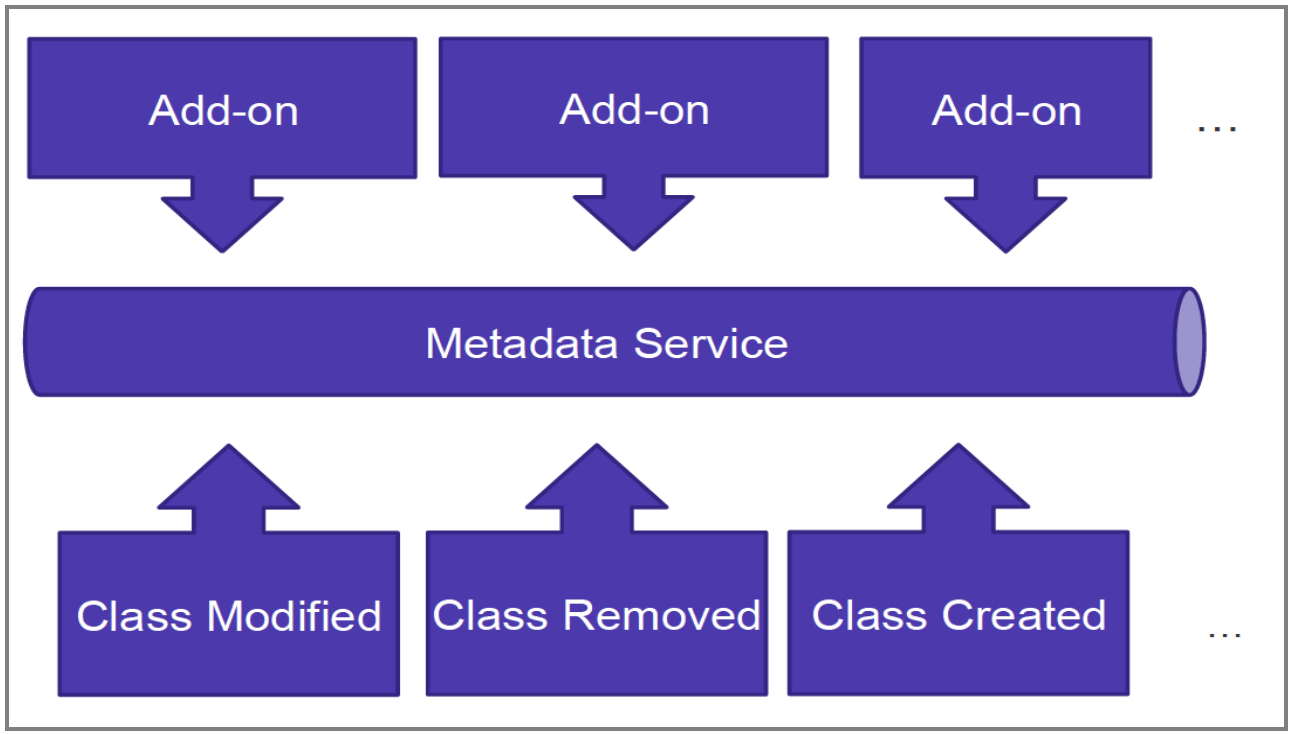
```
privileged aspect Course_Roo_ToString {

    public String Course.toString() {
        return ReflectionToStringBuilder
            .toString(this,
                      ToStringStyle.SHORT_PREFIX_STYLE);
    }
}
```

- Generated by shell via `@RooToString` on entity
- Generation done via ToString add-on

Injecting Commands



Roo Metadata Service



Why write addons?

- Replace the Maven archetype system
- Provide implementation patterns in live projects
- Generate and separate boilerplate code
- Achieve a RAD platform with your own code patterns
- Share with others – re-use of your own conventions

Tip: Build a Library of Recipe commands

- Your preferred ORM strategy / persistence mechanism
- Your web framework configuration
- Your set of standard libraries
- An Ajax approach
- Javascript, CSS, bootstrap libraries
- Anything you can think of...
- Share it with the outside world!

What are Roo Add-ons?

- Add-ons are OSGi bundles
- To understand this we have to detour
- What is OSGi in the context of Roo?

OSGi in 5 minutes (really)

- OSGi - dynamic module loading system
  - `Bundle` = Jar w/instructions (`META-INF/MANIFEST.MF`)
  - `OSGi Container` - a runtime platform (Apache Felix)
  - `OSGi Shell` - command line exposed by container
- Roo uses OSGi to mount and expose commands
- Roo does *not* use OSGi in production

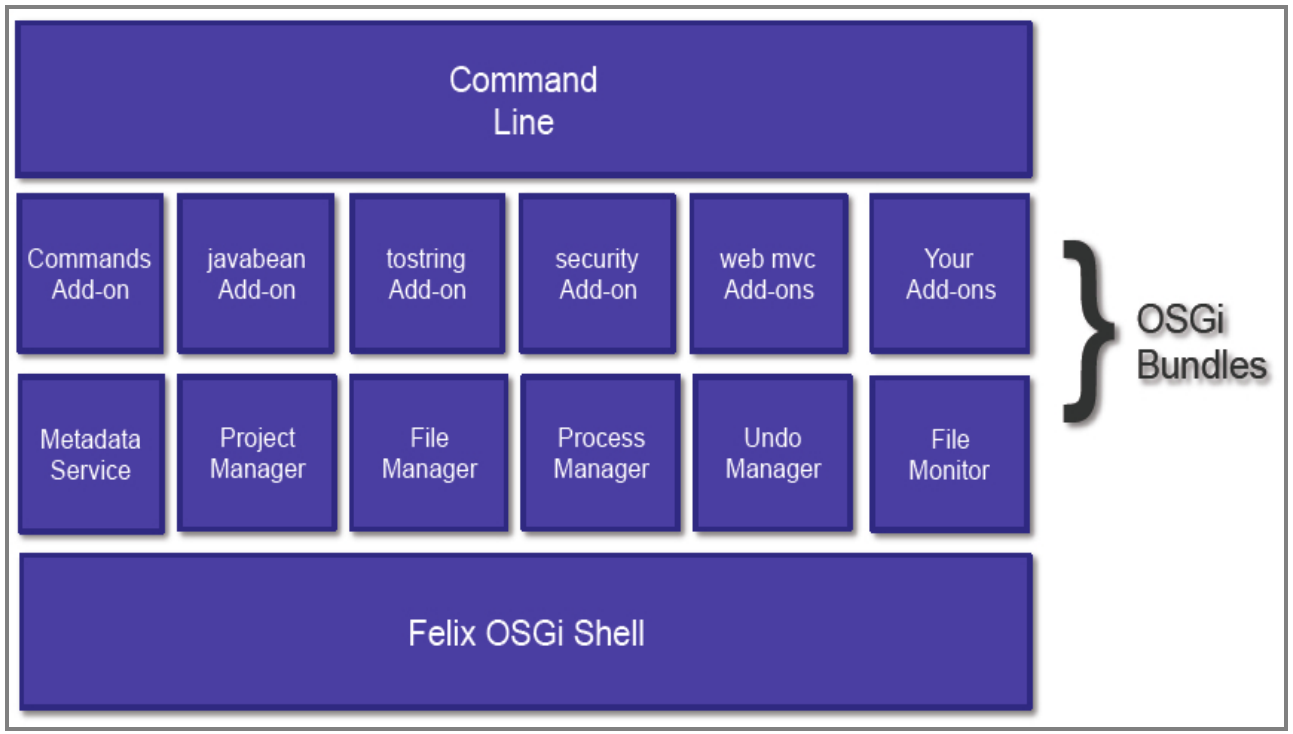
OSGi's dreaded `META-INF/MANIFEST.MF`

```
Manifest-Version: 1.0
Built-By: kenrimple
Tool: Bnd-1.15.0
Bundle-Name: jqueryui
Created-By: Apache Maven Bundle Plugin
Bundle-Copyright: Copyright Silly Weasel. All Rights Reserved.
Bundle-Vendor: Silly Weasel
Build-Jdk: 1.6.0_33
Bundle-Version: 1.0.1.RELEASE
Bnd-LastModified: 1343678841180
Bundle-ManifestVersion: 2
Bundle-License: http://www.apache.org/licenses/LICENSE-2.0.html
Bundle-Description: Installs jQuery 2% jeS
Import-Package:
org.apache.commons.io;version="[2.1,3)",org.springfram
ework.roo.classpath.operations;version="[1.2,2)",org.springframework.
roo.process.manager;version="[1.2,2)",org.springframework.roo.project
;version="[1.2,2)",org.springframework.roo.shell;version="[1.2,2)",or
g.springframework.roo.support.util;version="[1.2,2)",org.w3c.dom
Bundle-SymbolicName: org.sillyweasel.addons.jquery
Bundle-DocURL: http://www.rimple.com/roo-jquery-add-on/
```

Luckily, Roo does this FOR you using a Maven plugin...

You don't usually need to edit this file...

Roo and OSGi bundles



Simple add-ons - jQuery / jQuery UI

jQuery add-on features

- Installs the jQuery Javascript library
- Installs jQueryUI and associated images and theme files
- Exposed by two commands
  - `jquery setup` - sets up jQuery
  - `jqueryui setup` - installs jQuery UI

Creating the add-on

```
$ mkdir jquery-addon
$ cd jquery-addon
$ roo
...
roo> addon simple create
    --projectName jqueryaddon
    --topLevelPackage org.sillyweasel.addons
```

- Let's review the files Roo created...

Files created by the *simple* add-on

```
> tree
.
├── legal
│   └── LICENSE.TXT
├── log.roo
├── pom.xml
├── readme.txt
└── src
    └── main
        ├── assembly
        │   └── assembly.xml
        ├── java
        │   └── org
        │       └── sillyweasel
        │           └── addons
        │               ├── AddonsCommands.java
        │               ├── AddonsOperations.java
        │               ├── AddonsOperationsImpl.java
        │               └── AddonsPropertyName.java
        └── resources
            └── org
                └── sillyweasel
                    └── addons
                        ├── info.tagx
                        └── show.tagx

12 directories, 11 files
```

Let's do something simple...

Installing a simple command

- Expose a dummy `jquery setup` command
- Echo a success message to the Roo shell

Create a *command* OSGi Component

```
@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.");
   }

}
```

Run the example

```
roo> quit

$ mvn clean package

Build successful.

$ roo

roo> osgi start --url (full path to jar in ./target)
roo> jquery setup

jQuery installed.
```

Refactor - Delegate to another component

Add an Operations interface

```
package org.sillyweasel.jqueryaddon;
public interface JqueryaddonOperations {
    void setup();
}
```

- Interface driven components, just like Spring

The implementation class

```
package org.sillyweasel.addons.jquery;
...
@Component
@Service
public class JqueryOperationsImpl
  extends AbstractOperations
  implements JqueryOperations {

  @Reference
  private ProjectOperations projectOperations;

  public void setup() {...}

}
```

The `setup()` method in detail

```
public void setup() {

    String pathIdentifier =
       pathResolver.getFocusedIdentifier(
         Path.SRC_MAIN_WEBAPP, "js");

    copyDirectoryContents("jquery-1.8.1.min.js",
       pathIdentifier, true);

}
```

- `copyDirectoryContents` is a method of `AbstractOperations`
- We manually changed our bean to extend this class

Stage the asset(s)

```
src/main/resources *
          └── org
              └── sillyweasel
                  └── addons
                      └── jquery
                          └── jquery-1.8.1.min.js
```

Made available in JAR for copying into project

OSGi's SCR API makes delegation easy

- Expose components via `@Component` and `@Service`
  - `@Component` provides lifecycle
  - `@Service` provides component for injection
- Inject delegate with `@Reference`
- Sounds vaguely familiar...

Injection and calling `JqueryaddonOperations.setup()`

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

    @Reference
    private JqueryaddonOperations addonOperations;

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

What else can you do with Roo Components/Services

- Inject services from other bundles...
  - More complex with non--core-Roo components
  - Must wrestle with dependencies and/or MANIFEST entries
  - Not covered here - we will inject standard Roo services

Diversion - the Roo source code

(See http://github.com/springsource/spring-roo)

CLI command annotations

Key annotations

- `@CliAvailabilityIndicator` - Is the command (or list of commands) available?
- `@CliCommand` - Expose a command to the shell
- `@CliOption` - Expose command-line options for the given command

Example - Command with Parameters

```
@CliCommand(value = "coffeescript addjoinset", ...)
public void addJoinSet(

  // one for each option...
  @CliOption(
    key = "joinSetId",
    mandatory = true,
    specifiedDefaultValue = "main") String joinSetId,

  ...) {

   operations.addJoinSet(joinSetId, ...);

}
```

Picking sets of options

```
@CliCommand(value = "say hello",
   help = "Prints welcome message to the Roo shell")
public void sayHello(
  @CliOption(key = "name",
             mandatory = true,
             help = "State your name") String name,

  @CliOption(key = "countryOfOrigin",
             mandatory = false,
             help = "country") Country country) {...}
```

The `Country` enum

```
public enum Country {
   AUSTRALIA("Australia"),
   UNITED_STATES("United States"),
   GERMANY("Germany"), ...

   private String countryText;

   private Country(String value) {
        Validate.notBlank(propertyName,
           "Property name required");
        this.value = value;
   }
   public String toString() {
        return value;
   }
}
```

Dynamic attribute fill-in

Key Components

- Converters
  - Convert from text in the Roo Shell to Java types, *and*
  - Provides list of possible completion values, based on text entered so far
  - Similar to Spring converters, but not the same
- Target datatypes as Shell `@CliOption` param types
  - Built-in converters selected automatically
  - Your converter must be registered as a `@Component` and `@Service`

The Roo Converter interface

```
public interface Converter<T> {
  T convertFromText(String value, Class<?> targetType,
                    String optionContext);

  boolean getAllPossibleValues(
    List<Completion> completions,
    Class<?> targetType,
    String existingData,
    String optionContext,
    MethodTarget target);

  boolean supports(Class<?> type, String optionContext);
}
```

Built-in - BooleanConverter - `convertFromText`

```
public Boolean convertFromText(
  final String value,
  final Class<?> requiredType,
  final String optionContext) {

    if ("true".equalsIgnoreCase(value)
        || "1".equals(value)
        || "yes".equalsIgnoreCase(value)) {
        return true;
    }
    else if ("false".equalsIgnoreCase(value)
         || "0".equals(value)
         || "no".equalsIgnoreCase(value)) {
        return false;
    }
  ...
```

Dealing with type conversion issues

```
... else {
       throw new IllegalArgumentException(
          "Cannot convert " + value
          + " to type Boolean.");
    }
  }
```

- Exception will appear as output in the shell

Command Option Context

```
// in @CliOption
@CliOption(... optionContext = "path-a")

// in converter...
if (optionContext.equals("path-a")) {
   ... do something
}
```

- set context with `optionContext` on `@CliOption`
- converter can distinguish between multiple commands
- may need to convert differently based on the source

Converters built-in for

- Java Datatypes
  - `BigDecimal`, `BigInteger`, `Boolean`, `Character`, `Date`, `Double`, `Float`, `Integer`, `Long`, `Short`, `String`
- Files
  - The `FileConverter` can do completions based on files in the directory
- Other key converters
  - `Locale`, `Static Fields`, `Enum`
- Available Commands
  - via `AvailableCommandsConverter`

Example - PgpKeyIdConverter

```
@Component
@Service
public class PgpKeyIdConverter
  implements Converter<PgpKeyId> {

  @Reference
  private PgpService pgpService;

  public PgpKeyId convertFromText(
          final String value,
          final Class<?> requiredType,
          final String optionContext) {
      return new PgpKeyId(value.trim());
  }
  ...
```

Example - PgpKeyIdConverter - code fill-in

```
  public boolean getAllPossibleValues(...) {

    for (final PgpKeyId candidate :
            pgpService.getDiscoveredKeyIds()) {
        final String id = candidate.getId();
        if (id.toUpperCase().startsWith(
          originalUserInput.toUpperCase())) {
            completions.add(new Completion(id));
        }
    }
    return false; // can we dig deeper next time?
}
```

- The pgpService is an OSGi Roo service from the add-on

What about the Spring Shell Project?

Spring Shell

- A Shell programming interface for your own, Spring-based apps
- NOT OSGi based
- Turns out, you're learning it
- Uses `@CliCommand`, `@CliOption`, shell converters are the same
- Extracted from Spring Roo
- Now you know this too!

Advanced add-ons

Advanced add-on capabilities

- Configure the Maven POM
- Create, remove ITDs
- Respond to file create, update, removal events
- Modify existing Java class files
- Copy files (transactionally)

Configuring the Maven POM

- Use the `ProjectManager` Roo Service bean
- Can add, remove POM features
- POM wrapper objects for common Maven objects
  - `Dependency` - Maven dependencies
  - `Plugin` - Maven build plug-ins

Adding a dependency via the `ProjectManager`

```
// Direct API call
projectOperations.addDependency(
    projectOperations.getFocusedModuleName(),
    "cglib", "cglib-nodep", "2.2.2");
```

Adding multiple dependencies

```
Set<Dependency> dependencies = new HashSet<Dependency>();

// fill in from XML file, etc...

projectOperations.addDependencies(
    projectOperations.getFocusedModuleName(),
    dependencies);
```

Fetching dependencies from XML files

```
String focusedModuleName =
   projectOperations.getFocusedModuleName();

Element doc = XmlUtils.getConfiguration(getClass());

for (Element dependencyElement :
     XmlUtils.findElements(
       "/configuration/project/dependencies/dependency",
       doc)) {

  Dependency dependency =
    new Dependency(dependencyElement);

  projectOperations.addDependency(
    focusedModuleName, dependency);
}
```

Sample Configuration File

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration>
  <project>
    <dependencies>
      <dependency>
        <groupId>org.springframework.batch</groupId>
        <artifactId>
           spring-batch-admin-manager
        </artifactId>
         <version>1.0.0.RELEASE</version>
      </dependency>
    </dependencies>
  </project>
</configuration>
```

Other Maven POM sections

- You can add plug-ins, properties, dependency management
- Issues
  - The fetched POM wrapper objects are immutable
  - The constructors are long and a bit difficult
  - You may destroy existing user dependency/plugin settings
  - Consider a warning and a `--force` option for your users
  - Consider a `configuration.xml` file to simplify loading your new POM settings

Roo Ninja Skills

- Roo's typing and metadata system
- Adding annotations to Java classes
- Adding an ITD via detection of an annotation

Roo's Typing and Metadata System

Roo allows for Type Creation

- Add-ons can modify types and ITDs
- You need to provide:
  - A metadata object
  - A metadata provider object

Refining a Java Type

- Assumes a properly set-up add-on project
- Metadata class could extend `AbstractItdTypeDetailsProvidingMetadataItem`
- Allows for both ITD and class manipulation

In your meta-data class

```
protected AddonsMetadata(
  String identifier, JavaType aspectName,
  PhysicalTypeMetadata governorPhysicalTypeMetadata) {

  super(identifier, aspectName,
        governorPhysicalTypeMetadata);

itdTypeDetails = builder.build();
if (isValid()) {
    ensureGovernorExtends(
      new JavaType("java.lang.Thread"));
    ensureGovernorImplements(
      new JavaType("java.io.Serializable"));
    buildItd();
}
```

Adding ITD methods

- Use the `MethodMetadataBuilder` to build methods
- Add the builder you've created with `builder.addMethod`
- These methods get built into the Java class, not the ITD

Example ITD Method Builder

```
    private MethodMetadataBuilder createThreadSpawner() {
      final InvocableMemberBodyBuilder bodyBuilder =
        new InvocableMemberBodyBuilder();

      bodyBuilder.appendFormalLine(
        "System.out.println(\"invoking thread!\");");
      bodyBuilder.appendFormalLine(
        "new Thread().start(this);");
      bodyBuilder.appendFormalLine(
        "System.out.println(\"thread spawned!\");");

  ...
```

Example ITD Method Builder - returning the result

```
...

    return new MethodMetadataBuilder(
        getId(),
        Modifier.PUBLIC,
        new JavaSymbolName("sayHello"),
        JavaType.VOID_PRIMITIVE,
        bodyBuilder);
  }
```

- How do we call this?

The ITD Metadata's constructor

```
protected AsyncActionMetadata(...) {
  super(identifier, aspectName,
    governorPhysicalTypeMetadata);

  itdTypeDetails = builder.build();
  if (isValid()) {
    ...
  }
```

How to add the ITD

```
if (isValid()) {
    ensureGovernorImplements(
      new JavaType("java.lang.Runnable"));

    JavaType physicalType =
      governorPhysicalTypeMetadata.getType();

    MethodMetadataBuilder threadRunMethodBuilder =
        createThreadRunnerMethod(physicalType);

    builder.addMethod(threadSpawnerBuilder);
    buildItd();
  }
```

- Just add it as a builder

Type management in Roo is a huge topic

- Roo has a rich set of classes for manipulating types, ITDs
- Way beyond the scope of this talk
- Not documented except via JavaDocs
- Best to read the source code, experiment
- Any suggestions submit as JIRAs

Wrap-up

Roo add-ons can do almost anything

- Copy files
- Install dependencies
- Modify the POM file
- Configure / write XML
- Create / modify Java code
- Add/Remove/Manipulate ITDs

Questions

- Twitter: @krimple
- Blog: http://rimple.com
- Emerging Tech site: http://emergingtech.chariotsolutions.com