JBoss Forge Hands on Lab

Antonio Goncalves, Koen Aers, Ivan St. Ivanov, Daniel Cunha

Version 0.1
Oct 28, 2014

Table of Contents

Chapter 1. Introduction

1.1. JBoss Forge

It's not so easy to explain what Forge is in a few paragraphs. That is why we will re-use the introduction from Continuous Enterprise Development book:

If you've spent any time developing Java EE–based projects (or any nontrivial application, for that matter!), you've likely invested a good amount of energy in creating the project layout, defining dependencies, and informing the build system of the relevant class paths to be used in compilation and execution. Although Maven enables us to reduce that load as compared with undertaking project setup manually, there's typically quite a bit of boilerplate involved in the pom.xml defining your requirements.

JBoss Forge offers "incremental project enhancement for Java EE." Implemented as a command shell and integration with some IDE, Forge gives us the ability to alter project files and folders.

- Some concrete tasks we might use Forge to handle are:
    - Adding Java Persistence API (JPA) entities and describing their model
    - Configuring Maven dependencies
    - Setting up project scaffolding
    - Generating a view layer, reverse-engineered from a domain model
    - Deploying to an application server

Because Forge is built atop a modular, plug-in-based architecture, it's extensible to additional tasks that may be specific to your application. Overall, the goal of Forge is to ease project setup at all stages of development, so we'll be employing it in this text to speed along the construction of our examples.

1.2. What's this HoL about?

This hands-on lab (HoL) give you a good practical introduction to JBoss Forge. You will first install JBoss Forge, use it and then create addons to extend the capabilities of Forge. Forge can either by used with an IDE, or directly with a command line interface (CLI).

The idea is that you leave this hands on lab (HoL) with a good understanding of what JBoss is, what it is not, and how it can help you in your projects. Then, you'll have your entire time to investigate a big more and, hopefully, contribute ;o)

1.3. How does this HoL work?

You have this material in your hands (either electronically or printed) and you can now follow it step by step or choose any section "à la carte". The structure of this hands on lab is as follow :

- Installing Forge : in this section you will install JBoss Forge, either on a standalone mode, or with JBDS (JBoss Developer Studio)
- Using Forge : in this section you will play with Forge, create a project, add entities, scaffold a JSF and AngularJS web application, generate some Arquillian tests and deploy it on WildFly
- Developing Forge : in this section you will quickly create a full web application, build extensions, and add these extensions to the application

If Forge is already installed, just go to the "Using Forge" section ans tart using it. If you already know JBoss Forge a bit, jump to the section on "Developing Forge" and start hacking some addons. This "à la carte" mode allows you to make the most of this 3 hours long hands on lab.

1.4. What do you have to do?

This hands on lab should be as self explanatory as possible. So your job is to follow the instructions by yourself, do what you are suppose to do, and do not hesitate to ask for any precision, that's why the team is here. Make sure you have the needed software installed (see below) and be ready to get some fun.

1.5. Software requirements

The following software needs to be downloaded and installed:

- JDK 7
- Maven 3.2.x
- JBoss Forge 2.12.x
- WildFly 8.1

Chapter 2. Installing Forge

2.1. Installing Forge CLI

It's very simple install Forge CLI. Just download the ZIP to install Forge natively on your preferred Operation System.

2.1.1. Windows

On Windows, you can do:
Extract the distribution archive, to the directory you wish to install Forge.
Add the `FORGE_HOME` environment variable.

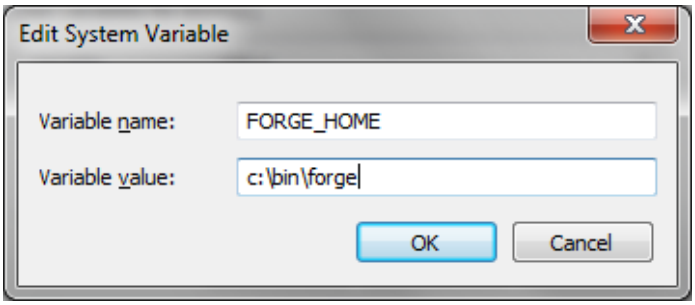

Figure 1. Installing Forge CLI Step 1

In the same dialog, add `%FORGE_HOME%\bin` to the system path.

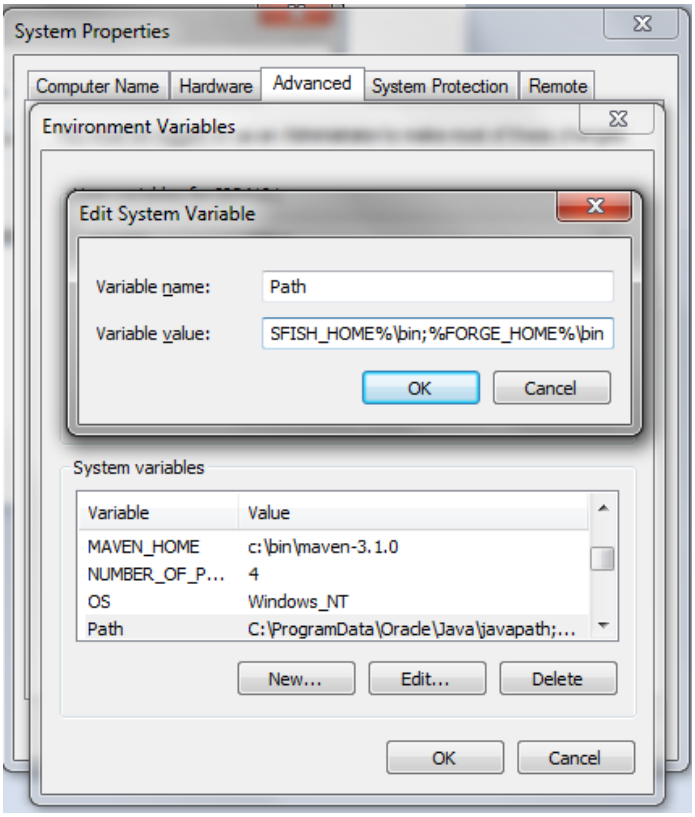

Figure 2. Installing Forge CLI Step 2

2.1.2. Mac OS X or Linux

On Mac OS X or Linux, you can do:
Extract the distribution archive, to the directory you wish to install Forge.
In a command terminal, add the `FORGE_HOME` environment variable, e.g:

```
export FORGE_HOME=/usr/local/jboss/forge-distribution-2.12.1.Final
```

Add FORGE_HOME/bin environment variable to your path, e.g:

```
export PATH=$FORGE_HOME/bin:$PATH.
```

or install via

```
curl http://forge.jboss.org/sh | sh
```

You have another alternative to install Forge on Mac OS X.
Use Homebrew to install Forge natively for use on the command-line, via:

```
brew install jboss-forge
```

2.2. Installing JBDS 8.0 with EAP

Installing JBDS is a piece of cake. Just download the installer from the JBoss website and in the target folder you issue:

```
java -jar jboss-devstudio-<version>-installer-eap.jar
```

This will launch the installation process for your platform. Windows, Linux and OSX are supported. The wizard will take you through a number of consecutive steps that are illustrated using screenshots below.

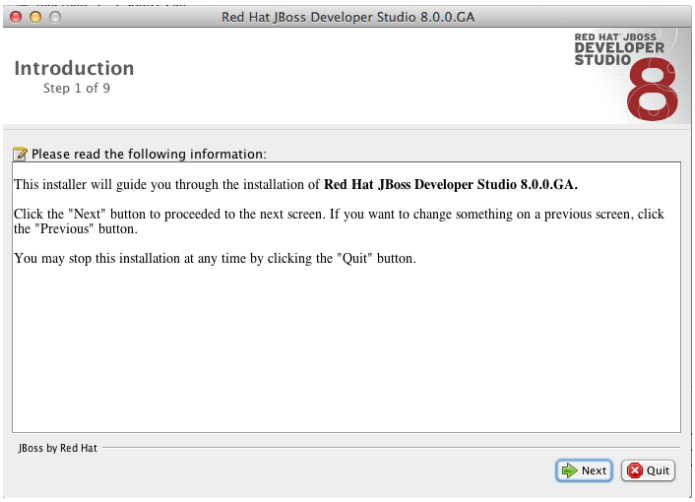

Figure 3. JBDS Installation Step 1

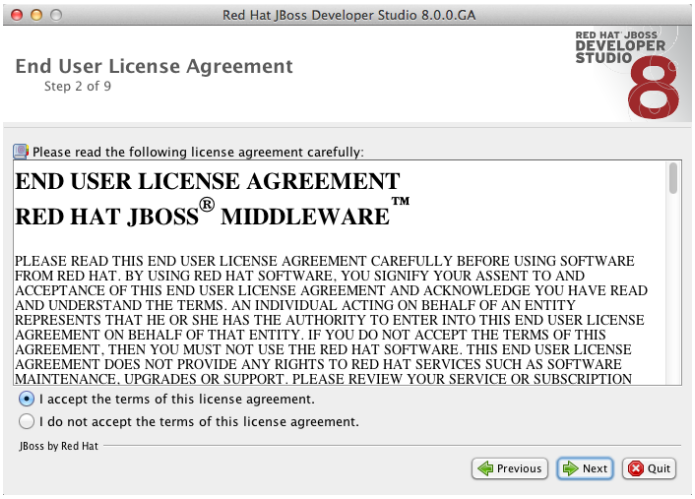

Figure 4. JBDS Installation Step 2

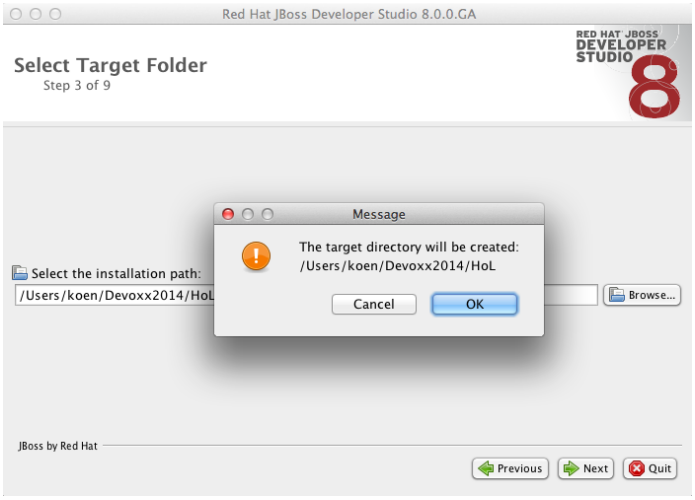

Figure 5. JBDS Installation Step 3

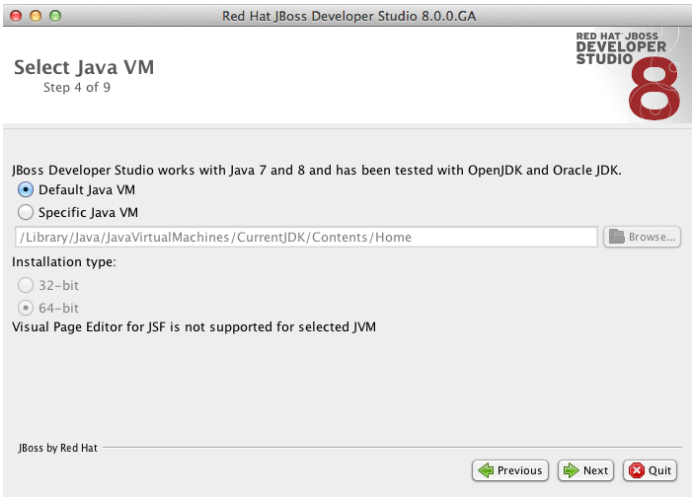

Figure 6. JBDS Installation Step 4

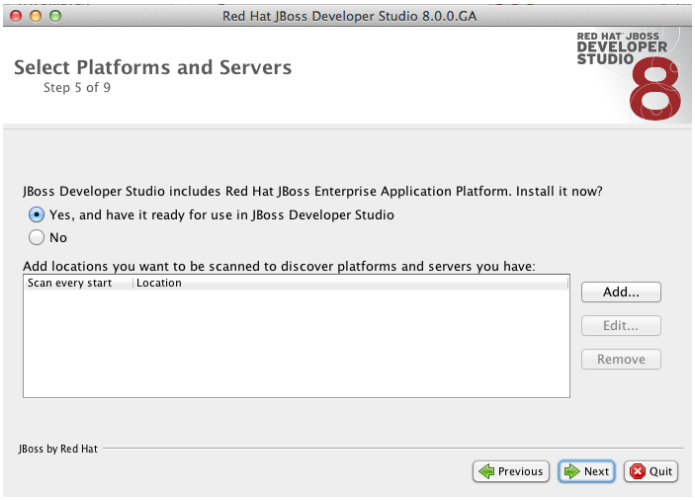

Figure 7. JBDS Installation Step 5

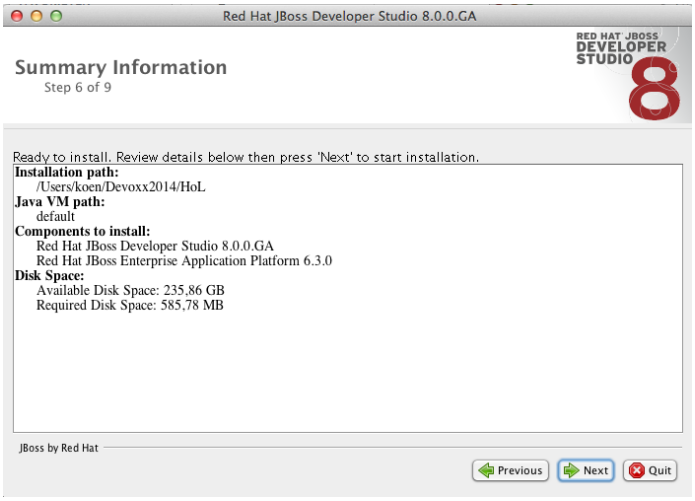

Figure 8. JBDS Installation Step 6

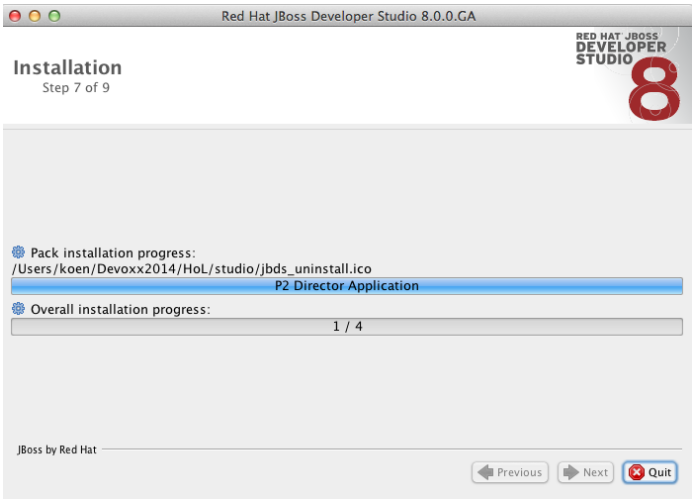

Figure 9. JBDS Installation Step 7

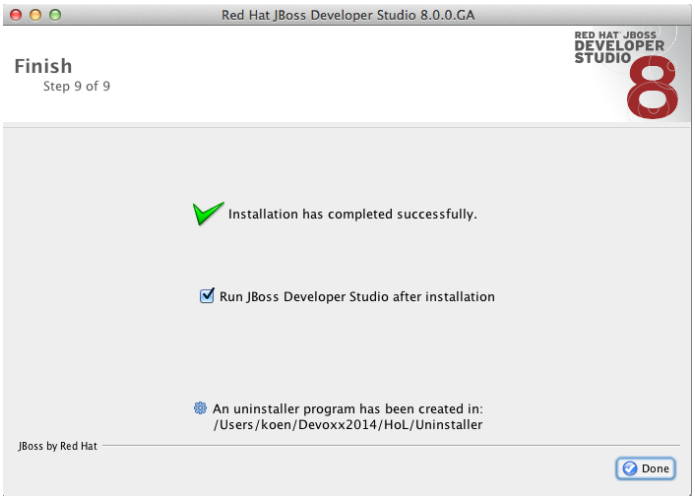

Figure 10. JBDS Installation Step 9

Congratulations! Now you are all set! Pressing `Done` will automatically launch JBDS.

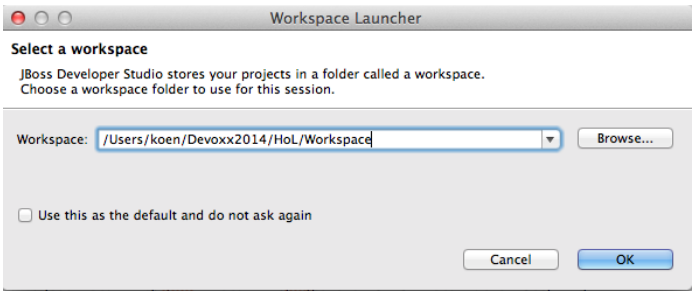

Figure 11. Choose Workspace

Choose an appropriate workspace and press `OK` to see JBDS in all its glory.

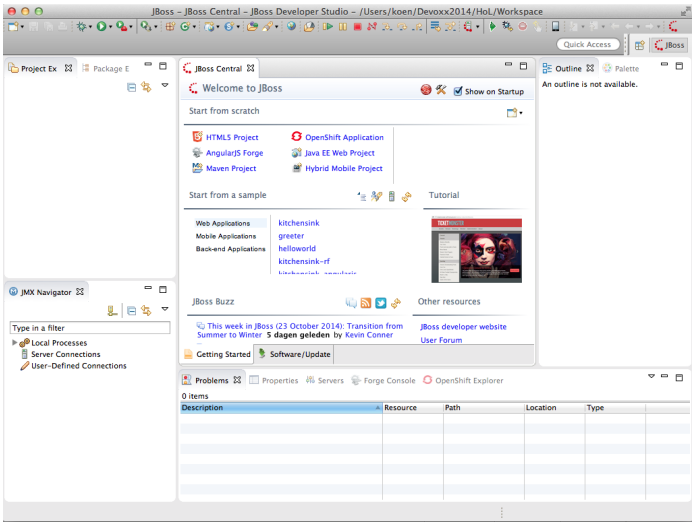

Figure 12. JBDS Welcome

Chapter 3. Using Forge

3.1. Creating a new project

Setting up a new project involves a lot of activities. You basically rely on a build and dependency management framework such as Maven and Gradle. Even if you feel comfortable reading the configuration files, i.e. pom.xml or build.gradle for those, it takes some time to write them from scratch. What you usually do is consulting manuals or textbooks, look in internet or most often - copy and paste them from one of your recent projects. Some of you may decide to use archetypes or IDE wizards, but you will soon realize that these generate too much garbage in your project configurate, that you will usually delete.

3.2. Setting up persistence and validation

3.3. Scaffolding JSF

3.4. Deploying on JBoss

3.5. Creating Arquillian tests

3.6. Scaffolding AngularJS

Chapter 4. Developing Forge

4.1. Developing a web application in few seconds

4.2. Developing Hibernate Envers addon

Hibernate Envers is a Hibernate core module that enables auditing of persistence classes. If you want to audit the history of all the changes made to a certain entity or one of its fields during the web application runtime, you just need to audit that with `@Audited`. Envers will create a separate table for each such entity, which will hold the changes made to it.

In this lab we will develop a Forge addon with the following features:

- Setup Envers for the following project by adding its dependency to the POM
- Enable auditing an entity by adding the `@Audited` annotation on class level

4.2.1. Creating a new Forge addon

Creating a new Forge addon is similar to any new project that you want to create. You can do it manually, you can copy and modify an existing project of the same type or you can use a wizard to do it for you. We would certainly recommend using Forge to help you bootstrap everything for several reasons. It knows what exactly which dependencies and artifacts you need as a start so you will not miss anything. Forge will also not create any garbage in your new project.

Before creating the Envers addon, you need to start Forge. Please make sure that you have followed the instructions in Installing Forge CLI before that. You can create a new addon if you run the following command in the Forge CLI:

```
project-new --named envers --type addon --topLevelPackage org.jboss.forge.addon --addons
org.jboss.forge.addon:javaee,2.12.0.Final
```

If you run Forge from JBDS, open the Forge wizard (Ctrl + 4 or CMD + 4 on Mac) then select *Project: New* and specify *envers* as project name, *org.jboss.forge.addon* as top level package, enter project location per your preference and as a Project type select *Forge Addon*:

This will create an empty Maven project that has the following artifacts:

- **pom.xml** where the top level package is the group ID and the project name is the artifact ID. Besides the minimum Forge dependencies, the command will add also those that you have specified with the `--addons` option in the format <group-id>:<artifact-id>,<version>
- **Standard maven directory structure** plus the top level package

- **Empty beans.xml** in the src/main/resources/META-INF directory. This is because Forge and its addons strongly rely on the CDI development model
- **README.asciidoc** file with a standard skeleton for documenting Forge addons

4.2.2. Developing the "Envers: Setup" command

The first command that we are going to create will set up Envers for a project. This basically means that the command will simply add the Envers library dependency to the current project POM. As with the new Forge addon, we can manually write the command class, copy and modify an existing command or let Forge itself generated it for us. Here we will go for the third option.

If you are running from the command line interface, type in:

```
addon-new-ui-command --named EnversSetupCommand --commandName "Envers: Setup"
--categories "Auditing"
```

While from the JBDS, after opening the Forge wizard (Ctrl + 4 or CMD + 4 on Mac), you should choose *Addon: New UI Command* and enter *EnversSetupCommand* in the Type Name field, *Envers: Setup* in the Command name field and add *Auditing* to the Categories list box:

This will generate `EnversSetupCommand` class in the `org.jboss.forge.addon.commands` package (unless you didn't specify explicitly anything else). Forge makes this class extend `AbstractUICommand`, which provides some basic functionality like configuring the command name, the command dialog and the command execution. We will go through these in this and the next few sections.

The `getMetadata()` method should be already implemented by Forge:

```
@Override
public UICommandMetadata getMetadata(UIContext context)
{
   return Metadata.forCommand(EnversSetupCommand.class).name(
         "Envers: Setup").category(Categories.create("Auditing"));
}
```

This will basically create a command that can be called *envers-setup* from the CLI (note the substitution of colons and spaces by hyphens) and as *Envers: Setup* in the *Auditing* category in the Forge wizard:

As the newly created command will not require any input from the user, we will leave the `initializeUI` method empty. However, in order to implement the command execution, we will need to change a little bit our class. More precisely we will have to extend from another abstract command class. The rationale behind this is that we want to update the **current** project POM. Extending `AbstractProjectCommand` instead of `AbstractUICommand` will give us some handy methods to access and manipulate the project configuration:

```
public class EnversSetupCommand extends AbstractProjectCommand
{
```

We will have to implement two more abstract methods coming from this parent class:

```
   @Override
   protected boolean isProjectRequired()
   {
      return true;
   }

   @Inject
   private ProjectFactory projectFactory;

   @Override
   protected ProjectFactory getProjectFactory()
   {
       return projectFactory;
   }
```

After having specified *Envers: Setup* as a project command, we can proceed to implementing the `execute` method. Usually this is called when the user clicks Finish on the command dialog or in our case where we don't require input: when the user selects the command from the Forge wizard.

As we mentioned earlier, the command will have to add the Hibernate Envers dependency to the project. We are going to build the Forge representation of this dependency using the DependencyBuilder's utility methods:

```
   @Override
   public Result execute(UIExecutionContext context) throws Exception
   {
      Dependency dependency =
            DependencyBuilder.create("org.hibernate")
                             .setArtifactId("hibernate-envers")
                             .setVersion("4.3.6.Final")
                             .setScopeType("provided");
   }
```

Speaking in Maven terms, this is a dependency to artifact with ID hibernate-envers, coming from the org.hibernate group, having version 4.3.6.Final and going into the project's *provided* scope.

After we have specified our dependency, we will have to add it to the project model. For that purpose we will use the `DependencyInstaller` utility, coming from the projects addon:

```
@Inject
private DependencyInstaller dependencyInstaller;
```

Forge 2.0 is based on modular runtime called *Furnace*. The core of Furnace itself is not bound to any development model, so the addons can decide which of the Furnace implementations it wants to use. We created our addon with the default configuration which enables the CDI development model. That is why we asked in the code snippet above Forge to provide us with the dependency installer for the current project build system.

Now it is time to install our dependency:

```
@Override
public Result execute(UIExecutionContext context) throws Exception
{
   Dependency dependency =
         DependencyBuilder.create("org.hibernate")
                          .setArtifactId("hibernate-envers")
                          .setVersion("4.3.6.Final")
                          .setScopeType("provided");
   dependencyInstaller.install(getSelectedProject(context), dependency);

}
```

We are using here one of the helper methods provided by the `AbstractProjectCommand`: `getSelectedProject()`.

Now our job is done, so it is time to report what we did. We do it by returning the result:

```
@Override
public Result execute(UIExecutionContext context) throws Exception
{
   Dependency dependency =
         DependencyBuilder.create("org.hibernate")
                          .setArtifactId("hibernate-envers")
                          .setVersion("4.3.6.Final")
                          .setScopeType("provided");
   dependencyInstaller.install(getSelectedProject(context), dependency);
   return Results.success("Envers was successfully setup for the current project!");
}
```

This will result in a SUCCESS: message in the command line interface and a green popup in the JDBS after our command is executed.

Now that we have a command the enables Hibernate Envers, it is time to add another command that

will turn on auditing for a given JPA entity.

4.2.3. Developing the "Envers: Audit entity" command

We will create the class for the new command in the same way that we created the one for "Envers: Setup": with the help of Forge. If you are running the CLI, then simply type:

```
addon-new-ui-command --named EnversAuditEntityCommand --commandName "Envers: Audit
entity" --categories "Auditing"
```

Or alternatively in the JBDS choose *Addon: New UI Command*, enter *EnversAuditEntityCommand* in the Type Name field, *Envers: Audit entity* in the Command name field and add *Auditing* to the Categories list box:

Then open the newly created class and make it extend `AbstractProjectCommand` instead of `AbstractUICommand` and also add the unimplemented methods the way you did it in the setup command.

This command will have to receive as input the entity class that has to be audited. To achieve this, we need to do two things:

1. Obtain and configure a `UIInput` object from Furnace
2. Add our input to the `UIBuilder` in the `initializeUI` method

Starting from number one, we should add the following member field to our command class:

```
    @Inject
    @WithAttributes(label = "Entity to audit", required = true)
    private UIInput<JavaResource> auditEntity;
```

Here we call our field auditEntity. This automatically will add a `--auditEntity` option to our command in the CLI. The type of the field is `UIInput<JavaResource>`, which means a few things:

- The JBDS integration will create a text box control for the audit entity, while the command line interface will expect a single unbounded value
- The type of the value for this option should be a file that represents a Java type (class, interface or enumeration)

We have also specified some additional attributes with the `@WithAttributes` annotation:

- The `label` attribute tells Forge's JBDS integration to override the field name (`auditEntity` in this case) with *Entity to audit*. This will be the actual label of the text box in the IDE. This will not however change the option name on the command line

- The `required` attribute will not let the user complete the dialog without entering a value for the entity. The well known asterisk character will be displayed along the label in JBDS

After we defined the input field, it is time to add it to the command dialog. In order to do that, we should edit the `initializeUI` method:

```
@Override
public void initializeUI(UIBuilder builder) throws Exception
{
   builder.add(auditEntity);
}
```

We can tell now Forge to show a *Browse* button to the right of the input field, which will open the well known type picker of Eclipse:

```
@Override
public void initializeUI(UIBuilder builder) throws Exception
{
   auditEntity.getFacet(HintsFacet.class).setInputType(InputType.JAVA_CLASS_PICKER);
   builder.add(auditEntity);
}
```

In Forge you can also set default values for a certain input. This way you can omit specifying its value on the command line and in the IDE it will be pre-filled in the command dialog. You can do that with the `setDefaultValue` method of the `UIInput`. In our case the UIInput is generified over the JavaResource class. So we'll have to check whether the current selection in the UI (being the CLI or JBDS) is a file that represents a Java type. If yes, we will set it as the default value of the text field:

```
@Override
public void initializeUI(UIBuilder builder) throws Exception
{
   auditEntity.getFacet(HintsFacet.class).setInputType(InputType.JAVA_CLASS_PICKER);
   Object selection = builder.getUIContext().getInitialSelection().get();
   if (selection instanceof JavaResource)
      auditEntity.setDefaultValue((JavaResource) selection);
   builder.add(auditEntity);
}
```

Now the UI of the command is ready. We can go on and implement the `execute` method. First we should get the value entered in the text field and convert it to `JavaResource`. Then we will extract the `JavaClassSource` out of it so that we can manipulate things like annotations:

```
@Override
public Result execute(UIExecutionContext context) throws Exception
{
   JavaResource javaResource = auditEntity.getValue().reify(JavaResource.class);
   JavaClassSource javaClass = javaResource.getJavaType();

}
```

Next we will check whether the chosen class has already the `Audited` annotation and if not, will add it to that. At the end we'll save the new content and will return successful result:

```
@Override
public Result execute(UIExecutionContext context) throws Exception
{
   JavaResource javaResource = auditEntity.getValue().reify(JavaResource.class);
   JavaClassSource javaClass = javaResource.getJavaType();
   if (!javaClass.hasAnnotation("org.hibernate.envers.Audited")) {
      javaClass.addAnnotation("org.hibernate.envers.Audited");
   }
   javaResource.setContents(javaClass);
   return Results.success(
       "Entity " + javaClass.getQualifiedName() + " was successfully audited");
}
```

But what if the user enters invalid input? This could be a file that does not exist, or is not a class or is not a JPA entity. We'll implement the `validate(UIValidationContext validator)` method to handle such situations. Whenever it finds illegal input, it will add a validation error to the `validator` parameter. This will bring an error message if the command executes in the CLI and in JBDS will disable the Finish button of the dialog, showing the error message in its well known location. This is how we implement the method:

```
@Override
public void validate(UIValidationContext validator)
{
   super.validate(validator);
   try
   {
      if (!auditEntity.getValue().reify(JavaResource.class).getJavaType()
            .hasAnnotation(Entity.class))
      {
         validator.addValidationError(auditEntity,
               "The selected class has to be JPA entity");
      }
   }
   catch (FileNotFoundException e)
   {
      validator.addValidationError(auditEntity,
            "You must select existing JPA entity to audit");
   }
}
```

Finally, we want to avoid some compilation errors in the project where we will run this command. So it should be only available for execution if the user has called the setup command first, i.e. if the current project has dependency to Hibernate Envers. You can implement this enabling and disabling in several ways. We will show one of these: by implementing the `isEnabled` method. There we will again obtain the `DependencyFacet` and will ask it whether the desired dependency is installed. If this method returns false, the Forge commands wizard will not list the Audit entity command and it will not be available in the command completion in CLI. This is the implementation:

```
@Override
public boolean isEnabled(UIContext context)
{
   Dependency dependency = DependencyBuilder
                           .create("org.hibernate")
                           .setArtifactId("hibernate-envers")
   return getSelectedProject(context).getFacet(DependencyFacet.class)
            .hasEffectiveDependency(dependency);
}
```

Our first addon is ready. We can now build it, deploy it and run it on the Java EE project that we created in the beginning of this chapter.

4.2.4. Installing and trying the Envers addon

4.3. Developing Logger addon

Chapter 5. Appendix

5.1. Acknowledgements

The following JBoss Forge community members have contributed, one way or another, to this hands-on lab:

- Antonio Goncalves (@agoncal)
- Daniel Cunha (@dvlc_)
- George Gastaldi (@gegastaldi)
- Ivan St. Ivanov (@ivan_stefanov)
- Koen Aers (@koentsje)
- Lincoln Baxter III (@lincolnthree)

5.2. Revision History

5.2.1. Version 0.1

- Creation of this material
- Using JBoss Forge 2.12.x
- Used during the conference Devoxx Belgium 2014
- JIRA https://issues.jboss.org/browse/FORGE-2102

5.3. Material

- The latest version of this document can be downloaded from https://github.com/forge/docs/tree/master/tutorials/forge-hol
- The PDF version is at https://github.com/forge/docs/blob/master/tutorials/forge-hol/docs/forge-hol.pdf

5.4. CLI Commands

Category	Command	Comment
Configuration	config-clear	

Configuration	config-list	
Configuration	config-set	
Database/Connections	connection-create-profile	Command to create a database connection profile.
Database/Connections	connection-remove-profile	Command to remove a database connection profile.
Forge/Generate	addon-new-annotated-ui-command	Generates an annotated UICommand implementation
Forge/Generate	addon-new-test	Generates a Furnace test case for an addon
Forge/Generate	addon-new-ui-command	Generates a UICommand implementation
Forge/Generate	addon-test-setup	Add addon test setup to this project
Forge/Manage	addon-build-and-install	Command to build and install a Furnace 2 addon.
Forge/Manage	addon-install	Command to install a Furnace 2 addon.
Forge/Manage	addon-install-from-git	Command to build and install a Furnace 2 addon.
Forge/Manage	addon-list	Command to list all currently installed Addons.
Forge/Manage	addon-remove	Command to remove a Furnace 2 addon.
Forge/Setup	addon-add-dependency	Adds the provided addon as a dependency to the selected project
JPA	jpa-new-mapped-superclass	Creates a new Mapped Superclass
Java	java-add-annotation	Add annotation to class, property or method.
Java	java-generate-equals-and-hashcode	Generates equals and hashcode for the given class
Java	java-generate-getters-and-setters	Generates mutators and accessors for the given class
Java	java-new-annotation	Creates a new Java Annotation

Java	java-new-class	Creates a new Java Class
Java	java-new-enum	Creates a new Java Enum
Java	java-new-enum-const	Creates a new Java Enum constant
Java	java-new-field	Creates a new field
Java	java-new-interface	Creates a new Java Interface
Java EE	javaee-setup	Setup Java EE in your project
Java EE/Bean Validation	constraint-add	Add a Bean Validation constraint
Java EE/Bean Validation	constraint-setup	Setup Bean Validation in your project
Java EE/Bean Validation	constraint-new-annotation	Create a Bean Validation constraint annotation
Java EE/Bean Validation	constraint-new-group	Create a Bean Validation group
Java EE/CDI	cdi-list-alternatives	
Java EE/CDI	cdi-list-decorators	
Java EE/CDI	cdi-list-interceptors	
Java EE/CDI	cdi-new-producer-field	Creates a new producer field
Java EE/CDI	cdi-setup	Setup CDI in your project
Java EE/CDI	cdi-new-bean	Creates a new CDI Managed bean
Java EE/CDI	cdi-new-conversation	Creates a conversation block in the specified method
Java EE/CDI	cdi-new-decorator	Creates a new CDI Decorator
Java EE/CDI	cdi-new-interceptor	Creates a new CDI Interceptor
Java EE/CDI	cdi-new-interceptor-binding	Creates a new CDI Interceptor Binding annotation
Java EE/CDI	cdi-new-qualifier	Creates a new CDI Qualifier annotation
Java EE/CDI	cdi-new-scope	Creates a new CDI Scope annotation
Java EE/CDI	cdi-new-stereotype	Creates a new CDI Stereotype annotation
Java EE/EJB	ejb-new-bean	Create a new EJB

Java EE/EJB	ejb-set-class-transaction-attribute	Set the transaction type of a given EJB
Java EE/EJB	ejb-set-method-transaction-attribute	Set the transaction type of a given EJB method
Java EE/EJB	ejb-setup	Setup EJB in your project
Java EE/JAX-RS	rest-generate-endpoints-from-entities	Generate REST endpoints from JPA entities
Java EE/JAX-RS	rest-new-cross-origin-resource-sharing-filter	Generate a Cross Origin Resource Sharing Filter
Java EE/JAX-RS	rest-setup	Setup REST in your project
Java EE/JAX-WS	soap-setup	Setup JAX-WS (SOAP) in your project
Java EE/JMS	jms-setup	Setup JMS in your project
Java EE/JPA	jpa-generate-daos-from-entities	Generate DAOs from JPA entities
Java EE/JPA	jpa-generate-entities-from-tables	Command to generate Java EE entities from database tables.
Java EE/JPA	jpa-new-embeddable	Create a new JPA Embeddable
Java EE/JPA	jpa-new-entity	Create a new JPA Entity
Java EE/JPA	jpa-new-entity-listener	Create a new JPA Entity Listener
Java EE/JPA	jpa-new-field	Create a new field
Java EE/JPA	jpa-setup	Setup JPA in your project
Java EE/JSF	faces-new-bean	Create a new JSF Backing Bean
Java EE/JSF	faces-new-converter	Create a new JSF Converter Type
Java EE/JSF	faces-new-validator	Create a new JSF Validator Type
Java EE/JSF	faces-new-validator-method	Create a new JSF validator method
Java EE/JSF	faces-set-project-stage	Set the project stage of this JSF project
Java EE/JSF	faces-setup	Setup JavaServer Faces in your project
Java EE/JSTL	jstl-setup	Setup JSTL in your project
Java EE/JTA	jta-setup	Setup JTA in your project
Java EE/Servlet	servlet-setup	Setup Servlet API in your project

Java EE/WebSocket	websocket-setup	Setup WebSocket API in your project
Java/ServiceLoader	service-register-as-serviceloader	Register a Java type as a service implementation.
Maven	archetype-add	
Maven	archetype-list	
Maven	archetype-remove	
Project	project-list-facets	Lists the facets associated with the current project
Project/Build	build	Build this project
Project/Generation	project-new	Create a new project
Project/Manage	project-add-dependencies	Add one or more arguments to the current project.
Project/Manage	project-add-managed-dependencies	Add one or more managed dependencies to the current project.
Project/Manage	project-add-repository	Add a repository to the current project descriptor.
Project/Manage	project-has-dependencies	Check one or more arguments in the current project.
Project/Manage	project-has-managed-dependencies	Check one or more managed dependencies in the current project.
Project/Manage	project-remove-dependencies	Remove one or more arguments from the current project.
Project/Manage	project-remove-managed-dependencies	Remove one or more managed arguments from the current project.
Project/Manage	project-remove-repository	Remove a repository configured in the current project descriptor.
Project/Manage	project-set-compiler-version	Set the java sources and the target compilation version
SCM / GIT	git-checkout	Checkout a branch from GIT repository or create a new one
SCM / GIT	git-clone	Clone a GIT repository
SCM / GIT	git-remove-pattern	Remove pattern from .gitignore

SCM / GIT	git-setup	Prepares the project for functioning in GIT context
SCM / GIT	gitignore-add-pattern	Add pattern to .gitignore
SCM / GIT	gitignore-create	Create .gitignore from templates
SCM / GIT	gitignore-edit	Open .gitignore and edit it
SCM / GIT	gitignore-list-patterns	List available .gitignore patterns
SCM / GIT	gitignore-list-templates	List all available .gitignore templates
SCM / GIT	gitignore-setup	Create .gitignore files based on template files from https://github.com/github/gitignore.git.
SCM / GIT	gitignore-update-templates	Update the local .gitignore template repository
Scaffold/Generate	scaffold-generate	Generates the scaffold
Scaffold/Setup	scaffold-setup	Setup the scaffold
Shell	cat	The cat utility reads files sequentially, writing them to the standard output. The file operands are processed in command-line order.
Shell	cd	Change the current directory
Shell	clear	Clear the console
Shell	cp	Copy a file or directory
Shell	echo	display a line of text
Shell	edit	Edit files with the default system editor
Shell	exit	Exit the shell
Shell	ls	List files
Shell	mkdir	Create a new directory.
Shell	open	Open files with the default system application
Shell	pwd	Print the full filename of the current working directory.
Shell	rm	Remove (unlink) the FILE(s).

Shell	run	Execute/run a forge script file.
Shell	touch	Create a new file or modify file timestamp.
Shell	track-changes	Initiate a transaction for each executed command.
Shell	transaction-commit	Commits a transaction
Shell	transaction-rollback	Rollbacks a transaction
Shell	transaction-start	Starts a transaction
Shell	wait	Wait for ENTER.
Shell	about	Display information about this forge.
Shell	command-list	List all available commands.
Shell	date	print current date
Shell	system-property-get	Get one or all system properties
Shell	system-property-set	Set a system property
Shell	version	Displays the current Forge version.
Shell	wait	Wait for ENTER.