

Spring Roo - Reference Documentation

DISID CORPORATION S.L.

Table of Contents

Getting started	1
1. Overview	2
2. What's new in Spring Roo 2.0	3
Improved extensibility	3
No backward compatibility	3
Usability improvements	3
Centered in Spring technologies	3
Application architecture	4
Domain model	4
View layer	4
3. Spring-Roo 2.0.0.RC2 New Features	6
Changes in the Roo Shell	6
Changes in generated applications	6
4. Requirements	8
5. Install Spring Roo	9
Using Spring Roo	11
6. The Roo shell	13
7. Impatient beginners	15
8. Create your Spring Boot application	16
9. Configure the project settings	17
10. Setup the persistence engine	18
11. The domain model	19
JPA entities	19
DTOs	23
12. The data access layer	24
Spring Data repositories	24
Default queries	24
13. The service layer	25
Service API and Impl	25
14. The view layer	26
Thymeleaf view engine	26
Spring MVC Controllers	26
Spring Webflow	28
15. The integration layer	29
REST API	29
WS API	29
Email	29
16. The security layer	31
Auditing JPA entities	31
17. The infrastructure layer	32
18. Customize the code	33
19. Javadoc in AsciiDoc	35

20. Running the application	36
Appendix	36
Appendix A: Command index for application development	37
!os	37
backup	38
cache setup	38
class	38
constructor	39
dto	40
email receiver setup	41
email sender setup	42
embeddable	44
entity jpa	44
entity projection	47
enum constant	49
enum type	49
equals	50
exit	51
field boolean	51
field date	53
field embedded	56
field enum	57
field file	59
field list	60
field number	64
field other	67
field reference	69
field set	72
field string	76
finder add	78
focus	79
help	79
hint	80
interface	80
jms receiver	81
jms sender	81
jpa audit add	82
jpa audit setup	83
jpa setup	84
module create	85
module focus	86
project setup	86
property add	87
property list	88

property remove	89
push-in	89
quit	90
repository jpa	91
script	92
security authorize	93
security filtering	93
security setup	95
service	95
settings add	97
settings list	98
settings remove	98
test integration	98
test unit	99
version	100
web flow	100
web mvc controller	101
web mvc detail	102
web mvc exception handler	104
web mvc finder	104
web mvc language	106
web mvc setup	107
web mvc templates setup	107
web mvc view setup	107
ws client	108
ws endpoint	109
Appendix B: Command index for add-on management	111
addon info bundle	111
addon install bundle	111
addon install url	111
addon list	112
addon remove	112
addon repository add	112
addon repository introspect	112
addon repository list	113
addon repository remove	113
addon search	113
addon suite install name	113
addon suite install url	114
addon suite list	114
addon suite start	114
addon suite stop	115
addon suite uninstall	115
Appendix C: Command index for add-on development	116

!g	116
addon create advanced	116
addon create i18n	117
addon create simple	117
addon create suite	118
addon create wrapper	119
addon development mode	120
metadata cache	120
metadata for id	120
metadata for module	120
metadata for type	121
metadata status	121
metadata trace	121
process manager debug	122
project scan now	122
project scan speed	122
project scan status	122
reference guide	123
system properties	123
Appendix D: Using Spring Roo without IDE	124
Installing Spring Roo	124
Backup and deployment	124
Appendix E: Roo Resources	126
Spring Roo Project Home Page	126
Downloads and Maven Repositories	126
StackOverFlow	126
Twitter	127
Issue Tracking	127
Source Repository	128
Commercial Products and Services	128
Other	129

2.0.0.RC2

© 2017 *The original authors.*

Copies of this document may be made for your own use and for distribution to others, provided that you do not charge any fee for such copies and further provided that each copy contains this Copyright Notice, whether distributed in print or electronically.

Getting started

Chapter 1. Overview

Spring Roo is an easy-to-use development tool for quickly building web applications in the Java programming language, which can be used as an standalone application or as an Eclipse or STS plugin. It allows you to build high-quality, high-performance, lock-in-free enterprise applications in just minutes.

What does it mean "Roo is a development tool"?

- **Roo isn't neither a library nor a framework.** Roo is not involved with your project when it runs in production. You won't find any Roo JARs in your runtime classpath. This is actually a wonderful thing. It means you have no lock-in to worry about. It also means there is no technical way possible for Roo to slow your project down at runtime, waste memory or bloat your deployment artefacts with JARs. We're really proud of the fact that Roo imposes no engineering trade-offs, as it was one of our central design objectives.
- **Roo is not an IDE plugin.** There is no requirement for a "Roo Eclipse plugin" or "Roo IntelliJ plugin". Roo works perfectly fine in its own operating system command window. It sits there and monitors your file system, intelligently and incrementally responding to changes as appropriate. This means you're perfectly able to use vi or emacs if you'd like (Roo doesn't mind how your project files get changed).
- **Roo is not an annotation processing library.** This allows Roo to work with a much more sophisticated and extensible internal model.

Best of all, Roo works alongside your existing Java and Spring knowledge, skills and experience. You probably will not need to learn anything new to use Roo, as there is no new language or runtime platform needed. You simply program in your normal Java way and Roo just works, sitting in the background taking care of the things you do not want to worry about.

Chapter 2. What's new in Spring Roo 2.0

Improved extensibility

Due to the OSGi container has been upgraded to OSGi R5, now Roo provides a new way to package and distribute a set of addons together: the Roo Addon Suite.

Roo Addon Suite is based on OSGi R5 Subsystems that provides a really convenient deployment model, without compromising the modularity of Roo.

No backward compatibility

Spring Roo 2.0 has important changes to achieve its goals, due to that, it contains API changes and less add-ons than previous version so **this release is not backward compatible with 1.x**.

It means Spring Roo 2.0 cannot neither update nor modify applications created with Spring Roo 1.x.

Usability improvements

The Spring Roo shell has improved its usability:

- More intuitive commands that provides only the necessary parameters.
- New commands to configure Spring Roo behavior.
- Maven multi-module support has been improved, now the intelligent `Ctrl+Space` (or `TAB`) completion will show you the applicable modules.
- New push-in commands for quicker and easier code customization.

Centered in Spring technologies

Now Spring Roo is centered in Spring technologies so addons like GWT addon and JSF addon have been moved to their own projects in order to be maintained by Roo community.

Moreover the generated applications are focused on newer Spring technologies like Spring IO platform, Spring Data, etc. Indeed, Spring Roo 2 creates Spring Boot applications.

Therefore, the XML configuration model has been replaced with the Java-based one.

Not only Spring

Most of the code generated by Roo is based on Spring technologies but not only on them, some parts of the application use other open source technologies, being the most important:

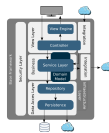
- [Apache CXF](#)
- [Springlets](#)
- ...

Application architecture

The architecture of the generated applications is based on commonly used patterns, like the *Separation of Concerns principle* and the *Domain Driven Design*.

There are hundreds of articles that explain the advantages of these patterns, but we would like to recommend:

- [Presentation Domain Data Layering](#), written by Martin Fowler.
- [Domain-Driven Design and Spring](#), by Oliver Gierke



The most notable improvements are:

- The default multimodule project set up the layers dependencies from top to bottom.
- Modularization based on generating both the API and the implementation.
- The Active Record data model has been removed in favor of Spring Data Repositories.

Domain model

- Improved entity relationship management: now Roo generates the needed logic to maintain the coherence of the relations taking in account the type of the relation, *Aggregation* or *Composition*.
- Added support and commands to generate DTO classes.

View layer

- Scaffold improvements:
 - Controllers refactored to support entity relationships management.
 - Master-detail view generation to manage the entity relations.
 - Several technologies for rendering views are supported. By default Spring Roo supports:

- Thymeleaf
- Jackson 2

Features of the Thymeleaf views:

- Dojo has been replaced with HTML5, CSS3, Bootstrap and jQuery components.
- They Include advanced UI components like [Select2](#) and [Datatables](#). The handler methods for those components (at controller classes) are also generated for easier customization.
- The Thymeleaf views include as few Javascript as possible by moving the Javascript code to *.js* files.
- View layer generation engine is based on Freemarker templates. Additionally Roo provides a command to install them in your project letting the ability to customize the view layer scaffold before executing it.
- New amazing Spring Roo Responsive Theme!

Chapter 3. Spring-Roo 2.0.0.RC2 New Features

The most important changes the new version of Spring Roo features regarding Roo 2.0.0.M3 are the following:

Changes in the Roo Shell

- New commands which allow:
 - Creating web flows in the generated applications.
 - Adding email senders and receivers.
 - Adding JMS senders and receivers.
 - Creating Web Services clients (WSDL) and endpoints (SEI).
- Improvements on test commands using latest Spring Boot Test features:
 - Improved data-on-demand generation to use in test commands.
 - Improved `test unit` command which now creates fully functional tests for JPA entities.
 - Improved `test integration` command which now allows integration test creation for JPA repositories and JSON/Thymeleaf controllers.
- Other command improvements:
 - Improved `field ...` commands readability, removing unnecessary options and make other dynamically visible.
 - Improved `push-in --method` command, allowing to distinguish between methods with same name but different arguments.
 - Bug fixes.

Changes in generated applications

- Generated code:
 - Improved read-only entities management.
 - Improved Deserializers generation.
 - Added default JavaDoc to all generated methods, constructors and fields.
 - Created `DataTables Advanced` extension to apply advanced configuration on `DataTables`

elements.

- Use Springlets URL generation.
- Bug fixes.
- View layer:
 - Added [datatables.mark.js](#) to mark the filtered text in DataTables component.
 - Added concurrency control in view layer.
 - Improved forms in view layer.
 - Integrated [JasperReports](#) with DataTables to allow exporting data to CSV, PDF and XLS.
 - Migrated Thymeleaf views to Thymeleaf 3.
 - Added new entity visualization support using `--entityFormatExpression` and `--entityFormatMessage` in some commands.
 - Added support for selecting those views of a particular entity which should show details.
 - Added new visualization system for one-to-one composition related entities.
 - Added multi-language support for Java enumerated constants.
 - Added support for multi row selection and batch delete.
 - Bug fixes.

Chapter 4. Requirements

To get started, please ensure you have the following system dependencies:

- A Linux, Apple or Windows-based operating system (other operating systems may work but are not guaranteed).
- A [Java JDK 6](#) or newer installed. Java **JDK 7** is recommended.
- [Apache Maven 3.3](#) or above installed and in the path.

We always recommend you use the latest version of Java and Maven that are available for your platform.

Chapter 5. Install Spring Roo

We recommend you use [Spring Tool Suite \(STS\)](#) which includes a number of features that make working with Spring Roo even easier (you can of course [use Roo without an IDE](#) at all if you prefer).

To install Spring Roo on your STS 3.8.2+ follow the instructions below:

1. [Java JDK 8](#) or newer is required.
2. Download the current release from Spring Roo project page [downloads section](#).
3. Unzip the distribution, which will unpack to a single installation directory; we will refer to it as `$ROO_HOME` from now on.
4. Go to [Spring Tool Suite™ Downloads](#) and follow the instructions to download and install the STS.

IMPORTANT

Sometimes, when use STS/Eclipse in Windows platform, there are difficulties while trying to use the JDK VM specified in the PATH. In that case, the solution is to modify the STS/Eclipse configuration by opening `STS.ini/Eclipse.ini` and adding the following lines **before** the `-vmargs` line:

- `-vm`
- `[JDK-DIR]/bin/javaw.exe`

(Don't put everything in a single line).

5. Open your STS IDE.
6. Install the Roo Extension from update site.

Because the release cycle of STS and Roo differ a version of Spring Roo may be in the Nightly or in the Release repository. This is not a problem, the installation process below will guide you which repository you should use depending on a given Roo version.

- i. Open **Help | Install New Software**.
- ii. Click **[Available Software sites]**.
- iii. Press the **[Import]** button.
- iv. Find the `"$ROO_HOME/conf/sts-sites-bookmarks.xml"` file and press **[OK]** button.
- v. Select the *Nightly* or *Release* site depending on the versions table below:

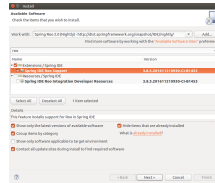
Version	STS update site
2.0.0.RC2	Spring Roo 2.0 (Nightly)

Version	STS update site
2.0.0.RC2	Spring Roo 2.0 (Nightly)
2.0.0.RELEASE	Spring Roo 2.0 (Release)

vi. Type the filter text *roo*

vii Select the feature **Spring IDE Roo Support**.

.



vii Press [**Next**]

i.

ix. Review the list of software that will be installed. Press [**Next**] again.

x. Review and accept licence agreement and press [**Finish**].

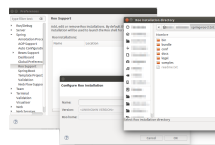
7. Restart the STS IDE

Configure Spring Roo 2.0.0

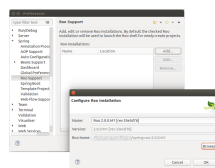
1. Open **Window | Preferences | Spring | Roo Support**.

2. In "*Roo Support*" press [**Add**] new installation button.

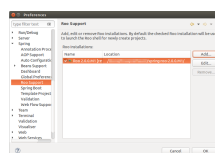
3. In "*Roo Configure Roo Installation*" press [**Browse**] button, then select the the directory in which Spring Roo 2.0.0 was unpacked, **\$ROO_HOME**.



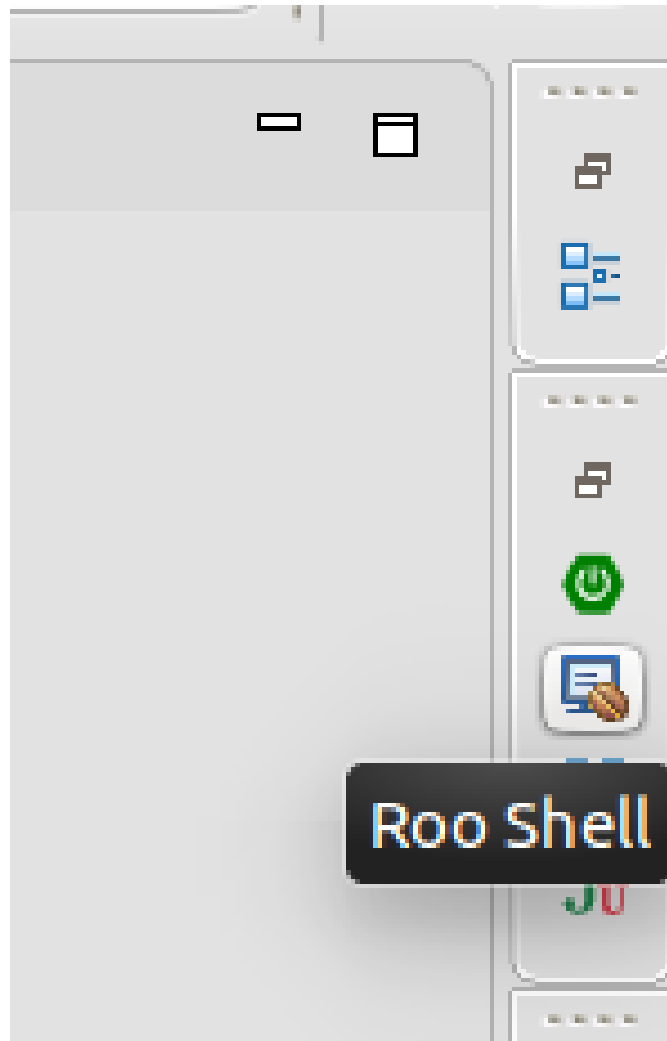
4. Confirm the new Roo installation.



5. Now Roo is installed in your STS.



6. Press [**Roo Shell**] button to open the Spring Roo Shell.



Using Spring Roo

The goal of this section is to familiarize you with the features of Spring Roo. For this purpose, we will build an application from scratch using Roo and following a domain-driven design philosophy.

In this project we're going to create the *Northwind* application in just ten minutes. This application is not a real application, which normally needs additional work, the goal is you understand how to use Spring Roo to create your own projects. To achieve that, we have designed this step-by-step guide to teach you almost all the Roo features.

The *Northwind* application is used by the employees of a fictitious company called Northwind Traders, which imports and exports goods from around the world.

We chose to build the sample application using Northwind because so many developers are already familiar with the domain of the problem. If you are not familiar with Northwind's domain, don't worry. It's a simple domain model with entites for Customers, Orders, Order Details, Products, etc.

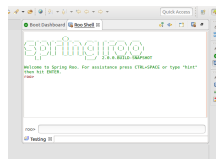
But first, let us to introduce the Roo shell.

Chapter 6. The Roo shell

The Spring Roo shell is an interactive shell that allows you to type *Roo* commands to perform code generation tasks.

Moreover by loading the "shell" in a window and leaving it running, as you make changes to your project, Roo intelligently determines what you're trying to do and takes care of doing it for you automatically. This usually involves automatically detecting file system changes you've made and then maintaining files in response.

We say "maintaining files" because Roo is fully round-trip aware. This means you can change any code you like, at any time and without telling Roo about it, yet Roo will intelligently and automatically deal with whatever changes need to be made in response. It might sound magical, but it isn't. This documentation will clearly explain how Roo works and you'll find yourself loving the approach - just like so the many other people who are already using Roo.



Here are some of the usability features that make the shell so nice to work with:

- *Tab completion*: The cornerstone of command-line usability is tab assist. Hit **Ctrl+Space** (or **TAB** if you're in a bash-like shell) and Roo will show you the applicable options.
- *Command hiding*: Command hiding will remove commands which do not make sense given the current context of your project. For example, if you're in an empty directory, you can type **project**, hit **Ctrl+Space**, and see the options for creating a project. But once you've created the project, the **project** command is no longer visible. The same applies for most Roo commands. This is nice as it means you only see commands which you can actually use right now. Of course, a full list of commands applicable to your version of Roo is available in the [command index appendix](#) and also via [help](#).
- *Option hiding*: Like command hiding, Roo will hide irrelevant command options for the current command context. This is specially useful for commands which have many options, giving the user an inside-command guidance by showing only the most important parameters for each moment of the command writing.
- *Contextual awareness*: Roo remembers the last Java type you are working with in your current shell session and automatically treats it as the argument to a command. You always know what Roo considers the current context because the shell prompt will indicate this just before it writes **roo>**.
- *Hinting*: Not sure what to do next? Just use the hint command. It's the perfect lightweight substitute for documentation if you're in a hurry!

Inbuilt help: If you'd like to know all the options available for a given command, use the help command. It lists every option directly within the shell.

- *Automatic inline help:* Of course, it's a bit of a pain to have to go to the trouble of typing help then hitting enter if you're in the middle of typing a command. That's why we offer inline help, which is automatically displayed whenever you press **Ctrl+Space** (or **TAB**). It is listed just before the completion options. To save screen space, we only list the inline help once for a given command option. So if you type `project --template` **Ctrl+Space** (or **TAB TAB TAB**), you'd see the inline help and the completion options
- *Scripting and script recording:* Save your Roo commands and play them again later.

You'll also have other neat Roo-IDE integration features, like the ability to press **Ctrl+R** (or **Apple+R** if you're on an Apple) and a popup will allow you to type a Roo command from anywhere within the IDE. Another nice feature is the shell message hotlinking, which means all shell messages emitted by Roo are actually links that you can click to open the corresponding file in an Eclipse editor.

There are two ways to work with Spring Roo:

1. Import existing Spring Roo projects. A simple import of the project using Eclipse's **File | Import | General | Maven Projects** menu option is sufficient.
2. Create new projects, as we will see in the next section.

Chapter 7. Impatient beginners

Spring Roo includes some examples to see it in action instantly.

If you are in a hurry to have an Spring Boot application up and running right away, execute one of the commands below:

The Northwind application (Maven multimodule project)

```
<strong>roo</strong> script --file northwind-multimodule.roo
```

Shop application with REST services

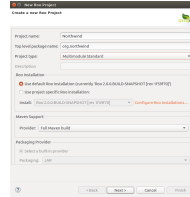
```
<strong>roo</strong> script --file restfulshop.roo
```

The classic Pet Clinic application (one Maven module project)

```
<strong>roo</strong> script --file clinic.roo
```

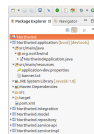
Chapter 8. Create your Spring Boot application

1. Open your STS IDE.
2. Open the **File | New | Spring Roo Project** wizard.



3. Fill the project data and press the [**Next >**] button. Then press [**Finish**].

Note we selected the *Multimodule Standard* project type, so Roo created you a Spring Boot & Maven multimodule project following the usual Maven-style directory structure:



For those familiar with Maven you will notice that this folder structure follows standard Maven conventions by creating separate folders for your main project resources and tests.

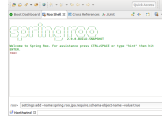
As you can see, the project extends the Spring IO platform, and it also adds the *spring boot starter* and the *spring boot starter test* dependencies.

Also Roo creates the Boot main application class.

Finally, both the parent pom and the modules pom files contain all required module dependencies, 3rd party dependencies and configurations to get started with the Northwind project.

Chapter 9. Configure the project settings

Project settings allows to set the configuration of some Roo commands. For example, in the [entity jpa](#) and [field](#) commands, the table and column names are optional, the **project settings** can modify this behaviour and set those parameters as mandatory so you don't forget to set the names.



Just type the Roo command on the right of the shell prompt, identified as **roo>**, and Roo will do the hard work.

In this example, disable it so you can go faster:

Set schema object names as optional

```
<strong>roo></strong> settings add --name spring.roo.jpa.require.schema-object-name  
--value false --force
```

NOTE

From now on we will illustrate the examples using commands in text format for easier test, just copying & pasting them in the STS Spring Roo shell.

Chapter 10. Setup the persistence engine

Once the project structure is created by Roo you can go ahead and install the data access layer configuration for your application.

Roo leverages the Spring Data JPA which provides a convenient abstraction to achieve object-relational mapping. JPA takes care of mappings between the persistent domain objects (entities) and their underlying database tables and Spring Data reduces the amount of boilerplate code required to implement the data access layer.

Execute the following command to configure the data access layer in the default Spring profile:

Setup data access layer

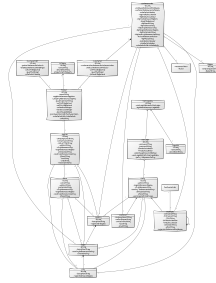
```
<strong>roo</strong> jpa setup --provider HIBERNATE --database HYPERSONIC_PERSISTENT
```

To change that configuration or to create another persistence configuration in a distinct Spring Profile you can use the `jpa setup` command as many times as needed. The command below will create another data access layer configuration in the `dev` profile:

Setup data access layer for dev profile

```
<strong>roo</strong> jpa setup --provider HIBERNATE --database H2_IN_MEMORY --profile  
dev
```

Chapter 11. The domain model



This class diagram represents a simplified model of the problem domain for the Northwind company, it is a good starting point for the application in order to deliver a first prototype.

JPA entities

Following the above class diagram, run the next commands to generate the Northwind domain entities:

1. Move to the module in which the model will be created:

```
<strong>roo</strong> module focus --moduleName model
```

2. Create the enums to use in the application:

Period, Status and Trimester enums

```
<strong>roo</strong> enum type --class ~.Period
enum constant --name QUARTERLY --class ~.Period
enum constant --name ANNUAL --class ~.Period

enum type --class ~.Status
enum constant --name NEWLY --class ~.Status
enum constant --name SEND_BILL --class ~.Status
enum constant --name SENT --class ~.Status
enum constant --name CLOSED --class ~.Status
enum constant --name CANCELED --class ~.Status

enum type --class ~.Trimester
enum constant --name FIRST_TRIM --class ~.Trimester
enum constant --name SECOND_TRIM --class ~.Trimester
enum constant --name THIRD_TRIM --class ~.Trimester
enum constant --name FOURTH_TRIM --class ~.Trimester
```

3. Create the entities:

Domain entities

```
<strong>roo</strong> entity jpa --class ~.City --readOnly  
    entity jpa --class ~.Country --readOnly  
    entity jpa --class ~.Region --readOnly  
    entity jpa --class ~.Category  
    entity jpa --class ~.OrderDetail  
    entity jpa --class ~.Party  
    entity jpa --class ~.PurchaseOrder  
    entity jpa --class ~.Report  
    entity jpa --class ~.Shipper  
    entity jpa --class ~.SoldProduct  
    entity jpa --class ~.Store  
    entity jpa --class ~.Supplier
```

Entity inheritance

```
<strong>roo</strong> entity jpa --class ~.Customer --extends ~.Party --force  
    entity jpa --class ~.Employee --extends ~.Party --force
```

Create the entities with special format when showing them in view layer. The format can be specified by a Spring Expression Language expression and also with a localized message (which can contain a SpEL too):

```
<strong>roo</strong> entity jpa --class ~.Product --entityFormatExpression "#{name}  
#{code}"  
    entity jpa --class ~.CustomerOrder --entityFormatMessage customerorder_format
```

4. Add the attributes to the entities:

Entity attributes and relationships

```
<strong>roo</strong> focus --class ~.Category
  field string --fieldName name
  field string --fieldName description
  field set --fieldName products --type ~.Product --mappedBy category

  focus --class ~.City
  field string --fieldName description
  field set --fieldName parties --type ~.Party --mappedBy city
  field set --fieldName customerOrders --type ~.CustomerOrder --mappedBy city
  field set --fieldName stores --type ~.Store --mappedBy city
  field set --fieldName suppliers --type ~.Supplier --mappedBy city

  focus --class ~.Country
  field string --fieldName description
  field set --fieldName parties --type ~.Party --mappedBy country
  field set --fieldName regions --type ~.Region --mappedBy country
  field set --fieldName customerOrders --type ~.CustomerOrder --mappedBy country
  field set --fieldName stores --type ~.Store --mappedBy country
  field set --fieldName suppliers --type ~.Supplier --mappedBy country

  focus --class ~.Customer
  field string --fieldName companyName
  field string --fieldName contactName
  field string --fieldName contactTitle
  field string --fieldName fax
  field string --fieldName email
  field set --fieldName customerOrders --type ~.CustomerOrder --mappedBy customer

  focus --class ~.CustomerOrder
  field date --fieldName orderDate --type java.util.Calendar --column ORDER_DATE
--persistenceType JPA_TIMESTAMP
  field date --fieldName requiredDate --type java.util.Calendar --persistenceType
JPA_TIMESTAMP
  field date --fieldName shippedDate --type java.util.Calendar --persistenceType
JPA_TIMESTAMP
  field number --fieldName freight --type java.math.BigDecimal
  field string --fieldName shipName
  field string --fieldName shipAddress
  field string --fieldName shipPostalCode
  field enum --fieldName status --type ~.Status --enumType STRING
  field string --fieldName shipPhone
  field date --fieldName invoiceDate --type java.util.Calendar --persistenceType
JPA_TIMESTAMP
  field date --fieldName closeDate --type java.util.Calendar --persistenceType
JPA_TIMESTAMP
  field set --fieldName orderDetails --type ~.OrderDetail --mappedBy customerOrder
```

```

    focus --class ~.Employee
    field string --fieldName firstName
    field string --fieldName lastName
    field string --fieldName title
    field date --fieldName birthDate --type java.util.Calendar --persistenceType
JPA_TIMESTAMP
    field date --fieldName hireDate --type java.util.Calendar --persistenceType
JPA_TIMESTAMP
    field string --fieldName extension
    field string --fieldName photo
    field string --fieldName notes
    field set --fieldName purchaseOrders --type ~.PurchaseOrder --mappedBy employee
    field set --fieldName customerOrders --type ~.CustomerOrder --mappedBy employee

    focus --class ~.OrderDetail
    field number --fieldName unitPrice --type java.math.BigDecimal
    field number --fieldName quantity --type java.lang.Integer
    field number --fieldName discount --type java.math.BigDecimal

    focus --class ~.Party
    field string --fieldName address
    field string --fieldName postalCode
    field string --fieldName phone

    focus --class ~.Product
    field string --fieldName name
    field string --fieldName code
    field string --fieldName quantityPerUnit
    field number --fieldName unitCost --type java.math.BigDecimal
    field number --fieldName unitPrice --type java.math.BigDecimal
    field number --fieldName unitsInStock --type java.lang.Integer
    field number --fieldName reorderLevel --type java.lang.Integer
    field other --fieldName discontinued --type java.lang.Boolean
    field set --fieldName purchaseOrders --type ~.PurchaseOrder --mappedBy product
    field set --fieldName orderDetails --type ~.OrderDetail --mappedBy product

    focus --class ~.PurchaseOrder
    field number --fieldName unitCost --type java.math.BigDecimal
    field number --fieldName quantity --type java.lang.Integer
    field date --fieldName orderDate --type java.util.Calendar --persistenceType
JPA_TIMESTAMP

    focus --class ~.Region
    field string --fieldName description
    field set --fieldName cities --type ~.City --mappedBy region
    field set --fieldName parties --type ~.Party --mappedBy region
    field set --fieldName customerOrders --type ~.CustomerOrder --mappedBy region

```

```

field set --fieldName stores --type ~.Store --mappedBy region
field set --fieldName suppliers --type ~.Supplier --mappedBy region

focus --class ~.Report
field string --fieldName type

focus --class ~.Shipper
field string --fieldName companyName
field string --fieldName phone
field set --fieldName customerOrders --type ~.CustomerOrder --mappedBy shipper

focus --class ~.Store
field string --fieldName name
field string --fieldName address
field string --fieldName postalCode
field string --fieldName phone

focus --class ~.Supplier
field string --fieldName companyName
field string --fieldName contactName
field string --fieldName contactTitle
field string --fieldName address
field string --fieldName postalCode
field string --fieldName phone
field string --fieldName fax
field string --fieldName web
field set --fieldName products --type ~.Product --mappedBy supplier
field set --fieldName stores --type ~.Store --cardinality MANY_TO_MANY

```

DTOs

DTOs (Data Transfer Objects)

```

<strong>roo</strong> dto --class ~.ShipperPhoneFormBean
    field string --fieldName phone

dto --class ~.CustomerOrderFormBean --serializable
field number --fieldName orderId --type java.lang.Long
field number --fieldName employeeId --type java.lang.Long
field number --fieldName customerId --type java.lang.Long
field date --fieldName orderDate --type java.util.Calendar
field string --fieldName employeeName
field string --fieldName customerCompanyName
field other --fieldName status --type ~.Status
field date --fieldName shippedDate --type java.util.Calendar
field number --fieldName freight --type java.math.BigDecimal

```

Chapter 12. The data access layer

Spring Data repositories

It is possible to specify an entity projection as a default return type for repository queries:

- Create the entity projection:

```
<strong>roo</strong> entity projection --class model:~.CustomerInfo --entity  
model:~.Customer --fields id,companyName,email,fax --entityFormatExpression  
#{companyName}
```

- Create the repository for the entity, which will use the projection as default return type of queries:

```
<strong>roo</strong> repository jpa --entity model:~.Customer --interface  
repository:~.CustomerRepository --defaultReturnType model:~.CustomerInfo
```

Create repositories for all the remaining entities:

```
<strong>roo</strong> repository jpa --all
```

Default queries

```
<strong>roo</strong> finder add --entity <strong>model:</strong>~.Shipper --name  
findByCompanyName  
    finder add --entity model:~.Region --name findByCountryIdOrderByDescriptionAsc  
    finder add --entity model:~.City --name findByRegionIdOrderByDescriptionAsc  
    finder add --entity model:~.Product --name findByDiscontinuedOrderByNameAsc  
    finder add --entity model:~.Shipper --name findByPhone --formBean  
model:~.ShipperPhoneFormBean
```

Since Spring Roo 2.0, the multimodule support lets to prefix the module name to the entity path to select the Maven module in which the new entity will be created. Spring Roo will propose the available module names when hit **Ctrl+Space** (or **TAB** if you're in a bash-like shell).

Chapter 13. The service layer

Service API and Impl

```
<strong>roo</strong> service --all
```

Chapter 14. The view layer

The Spring Roo Web MVC scaffolding can deliver a fully functional web frontend and REST API to your domain business logic. The scaffolding support allows you to scaffold Spring MVC controllers, Thymeleaf views and REST API for an existing domain model.

First of all, you must add the web support to the application. All needed updates in the project will be performed by Roo.

Setup the view layer

```
<strong>roo</strong> web mvc setup
```

Remember that now, Roo generates applications centered in Spring technologies, you will notice that the generated artifacts configure Spring MVC in your application.

In Spring Roo 2 the view layer generation system has been refactored to support several technologies for rendering views. Spring Roo 2 supports [Thymeleaf](#) and [Jackson](#).

Thymeleaf view engine

The `web mvc view setup` allows you to install and configure the artifacts that will let to scaffold a Thymeleaf based view layer.

```
<strong>roo</strong> web mvc view setup --type THYMELEAF
```

Optionally, you can tell Roo to copy the templates it uses to generate the view templates to the application's `.roo/templates/thymeleaf/` directory, allowing the developers to customize them for code generation:

Install the templates to generate the view templates

```
<strong>roo</strong> web mvc templates setup --type THYMELEAF
```

Spring Roo uses [Freemarker](#) templates for generating the Thymeleaf view templates, you will notice that the `.roo/templates/thymeleaf/` contains the `.ftl` files.

Spring MVC Controllers

The controller command will scaffold the given domain entity and it will create both the Spring MVC controllers and the templates to generate the view response .

Generate the views and controllers to manage the domain entities (CRUD)

```
<strong>roo</strong> web mvc controller --entity model:~.Category --responseType THYMELEAF
web mvc controller --entity model:~.Country --responseType THYMELEAF
web mvc controller --entity model:~.CustomerOrder --responseType THYMELEAF
web mvc controller --entity model:~.Customer --responseType THYMELEAF
web mvc controller --entity model:~.Employee --responseType THYMELEAF
web mvc controller --entity model:~.Product --responseType THYMELEAF
web mvc controller --entity model:~.Shipper --responseType THYMELEAF
web mvc controller --entity model:~.SoldProduct --responseType THYMELEAF
web mvc controller --entity model:~.Store --responseType THYMELEAF
web mvc controller --entity model:~.Supplier --responseType THYMELEAF
web mvc controller --entity model:~.City --responseType THYMELEAF
web mvc controller --entity model:~.Region --responseType THYMELEAF
web mvc controller --entity model:~.PurchaseOrder --responseType THYMELEAF
```

As you can see, since Spring Roo 2.0 the **web mvc controller** has the parameter **--responseType** that lets to indicate the rendering view technology to scaffold. You can chose one of the two available rendering view technologies:

- *JSON* (default), generate JSON messages using Jackson 2.
- *THYMELEAF*, generate HTML5 pages using Thymeleaf template engine.

Entity relationship management

You can generate master-detail views to manage the entity relations as follows:

Relationship controllers and views

```
<strong>roo</strong> web mvc detail --entity model:~.Category --field products
--responseType THYMELEAF --views list,show
web mvc detail --entity model:~.Product --field purchaseOrders --responseType THYMELEAF --views list,show
web mvc detail --entity model:~.Country --responseType THYMELEAF --field regions
--views list,show
web mvc detail --entity model:~.Region --responseType THYMELEAF --field cities
--views list,show
```

Search support

Finally, create the views to search entities.

Search controllers and views

```
<strong>roo</strong> web mvc finder --all --responseType THYMELEAF
```

Spring Webflow

CustomerOrder web flow

```
<strong>roo</strong> web flow --flowName customerOrdersFlow --class  
~.CustomerOrderFormBean
```

Chapter 15. The integration layer

Today, applications must necessarily connect to many types of external systems. Spring Roo generate the connectors to send data and the endpoints to receive information to and from those systems in the outside.

REST API

Spring Roo can create a full REST API to manage the entities. You only have to execute the command below and Roo will generate one Spring MVC REST controller for each entity.

REST services

```
<strong>roo</strong> web mvc controller --all --pathPrefix /api
```

Roo has generated the controllers with handler methods to create, update, delete single entities and collection of entities. In addition, the controllers will have methods to find data following the REST principles.

WS API

Spring Roo generate SOAP Services easily, available under `/services` URL.

WebServices

```
<strong>roo</strong> ws endpoint --service service-api:~.CategoryService --sei  
application:~.ws.api.CategoryWebService --class  
application:~.ws.endpoint.CategoryWebServiceEndpoint --config  
application:~.config.WsEndpointsConfiguration
```

Email

Send email

```
<strong>roo</strong> email sender setup --service service-impl:~.CustomerServiceImpl  
--username USERNAME --password PASSWORD --host HOST --port 1000 --protocol PROTOCOL  
--starttls true
```

Receive email

```
<strong>roo</strong> email receiver setup --service service-impl:~.EmployeeServiceImpl  
--username USERNAME --password PASSWORD --host HOST --port 1000 --protocol PROTOCOL  
--starttls true
```

Chapter 16. The security layer

Create and configure the Spring Security artifacts that will protect your application.

```
<strong>roo</strong> security setup --provider SPRINGLETS_JPA
```

As you can see, since Spring Roo 2.0 the `security setup` has the parameter `--provider` that will let to indicate which security provider will create the security artifacts.

A security provider is simply a configurer that will create and configure the security artifacts in its way.

Currently you can chose one of the two available providers:

- *DEFAULT*, configures the Spring Boot security defaults.
- *SPRINGLETTS_JPA*, sets the Spring Boot defaults plus the Springlets JPA authentication provider.

Now, grant the permissions that restricts executing the domain logic, for example, only the users with roles `ADMIN` or `EMPLOYEE` are granted to delete customers.

```
<strong>roo</strong> security authorize --class service-api:~.CustomerService --method  
delete --roles ADMIN,EMPLOYEE
```

Auditing JPA entities

Adds support for auditing a JPA entity. It will add the Spring Data JPA entity listener to capture auditing information on persiting and updating entities.

```
<strong>roo</strong> jpa audit setup  
jpa audit add --entity model:~.Category
```

Chapter 17. The infrastructure layer

By *infrastructure layer* we mean the layer that contains those project artifacts that aren't related directly with the problem domain, like tests, logging, etc.

```
<strong>roo</strong> test unit --class model:~.CustomerOrder
  test unit --class model:~.Category
  test unit --class model:~.Product

  test integration --class repository:~.CategoryRepository
  test integration --class repository:~.CityRepository
  test integration --class repository:~.CountryRepository
  test integration --class repository:~.CustomerOrderRepository
  test integration --class repository:~.CustomerRepository
  test integration --class repository:~.EmployeeRepository
  test integration --class repository:~.OrderDetailRepository
  test integration --class repository:~.PartyRepository
  test integration --class repository:~.ProductRepository
  test integration --class repository:~.PurchaseOrderRepository
  test integration --class repository:~.RegionRepository
  test integration --class repository:~.ReportRepository
  test integration --class repository:~.ShipperRepository
  test integration --class repository:~.SoldProductRepository
  test integration --class repository:~.StoreRepository
  test integration --class repository:~.SupplierRepository

  test integration --class
application:~.web.CustomerOrdersCollectionThymeleafController
  test integration --class application:~.web.CountriesItemRegionsThymeleafController
  test integration --class application:~.web.CategoriesItemThymeleafController
  test integration --class application:~.web.PurchaseOrdersItemJsonController
  test integration --class application:~.web.OrderDetailsCollectionJsonController
```

Chapter 18. Customize the code

You can easily modify the Roo-generated code by using the Eclipse/STS AJDT Refactoring Push-in feature.

The AJDT refactoring moves intertype declarations (methods, fields, etc) into their target types. From then, the method, field, etc. will be in the Java source file. Roo detects that change in the project and the declaration in the Java file will take priority over code generation so Roo won't re-generate it whereas the declaration is in the Java file.

To *push-in* the Roo-generated code:

1. Edit Java source file.
2. Open the [Cross References](#) view.

NOTE

If the Cross References view is empty you must re-build the project by executing **Project | Clean ...**. It occurs when the crosscutting information is missing, so you must re-build the project in order to re-generate the crosscutting information shown in the Cross References view.



3. Double click on the aspect declaration. The the ITD file is opened in the AspectJ/Java editor.
4. Right click on the aspect declaration, then run **AspectJ_Refactoring | Push In ...**
5. Finally re-build the project by executing **Project | Clean**.

At this point, the developer can modify the Java source file, Roo will not overwrite or modify any Java source file.

A quicker way to take the control of the generated code is using the **push-in** command. This command moves in batch, intertype declarations into the target type. For example you can move the classes in one package from the .aj file to the .java file executing one command only:

```
<strong>roo</strong> push-in --package model:org.northwind.model
```

In summary, you can easily modify the Roo-generated code by using the Eclipse/STS AJDT Push-in feature or by using the **push-in** command.

Project without .aj files

A simple way of stopping to use Roo is to simply never load it again. The **Roo**.aj files will still be on disk and your project will continue to work regardless of whether the Roo shell is never launched again. You can even uninstall the Roo system from your computer and your project will still work. The advantage of working in this way is that you have not lost the benefits of using Roo, and it is very easy to use Roo shell again in the future.

NOTE

Spring Roo needs that .aj files to maintain the generated code automatically. Is not possible to know which code has been generated by Spring Roo shell and which code has been modified by developers without the .aj files.

Anyway, if you don't want to have .aj files in your generated project, you could use the following command to make push-in of all the generated code:

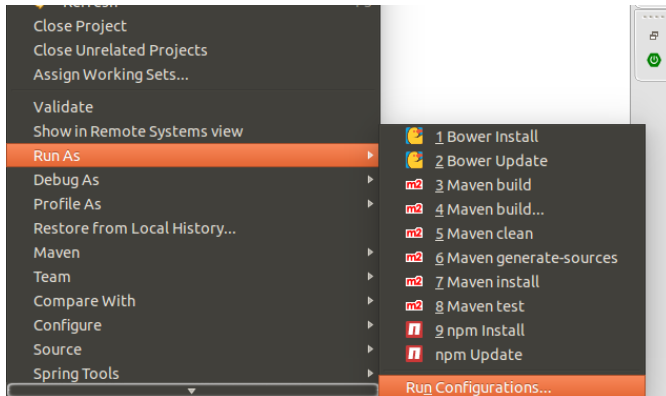
```
<strong>roo</strong> push-in --all --force
```

Chapter 19. Javadoc in AsciiDoc

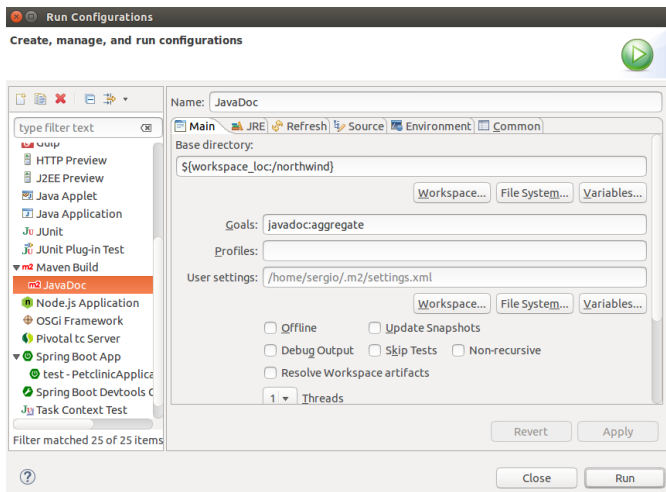
Spring Roo generated projects automatically include the "maven-javadoc-plugin" to generate project documentation following AsciiDoc syntax. This configuration it's done by using "[Asciidoclet](#)".

To generate the project's documentation you can follow the following steps:

1. Go to the STS "Package Explorer".
2. Right click in the project and go to **RunAs | Run Configurations...**



3. In the window that will open, double click in [**Maven Build**] item from submenu.
4. In the configuration window, specify **javadoc:aggregate** as Maven goal.
5. Set the project's root directory as "Base directory". You can easily do it by clicking *Workspace...* and selecting the root module of your project.

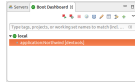


6. Apply configuration and close the window, or execute it directly with *Run*.
7. The generated JavaDoc will be in "[*ROOT-PROJECT*]/target/site/apidocs/".

Chapter 20. Running the application

You can deploy your project using "Boot Dashboard":

1. Go to the "*Boot Dashboard*" view.
2. Select the right module of your project, one of the modules that contain a class annotated with `@SpringBootApplication`. Then press **[Start]** button



3. The application should be available under the following URL <http://localhost:8080/Northwind>

Appendix

Appendix A: Command index for application development

- **Description:**

Commands are listed in alphabetic order, and are shown in monospaced font. with all the options you can specify when using the command. Most commands accept a large number of options, and all of the possible options for each command are presented in this appendix.

- **Syntax:**

The Roo command syntax is presented with some marks to easily distinguish the different parameters of the commands depending their behaviour in the shell:

- Mandatory: `{--parameter}`. This kind of parameter must always be provided when executing the command.
- Dynamic mandatory: `(--parameter)`. Depending on the project's context or the already provided parameters, this kind of parameter may be mandatory.
- Optional: `[--parameter]`. This kind of parameter is optional when executing the command.
- Dynamic optional: `([--parameter])`. This kind of parameter is optional, but it can only be used when an particular condition is fulfilled.
- Mutual excluding: `--parameter ... | --parameter ...`. Both groups of parameters in the two sides of the `|` are mutually selective. If one of them is specified, the other won't be available.

!os

Allows execution of operating system (OS) commands. Ex.: `!os mkdir test_dir`

```
<strong>roo</strong> !os [--command]
```

- *Optional:*

--command

The OS command to execute.

Default: "

backup

Backups your project to a zip file located in root directory.

```
<strong>roo</strong> backup
```

This command does not accept any options.

cache setup

Installs support for using intermediate memory in generated project by using Spring Cache abstraction. Users can specify different providers to use for managing it.

```
<strong>roo</strong> cache setup [--provider --profile]
```

- *Optional:*

--provider

Parameter that indicates the provider to use for managing intermediate memory. Possible values are: **GUAVA**.

--profile

Parameter that indicates the name of the profile that will be applied.

class

Creates a new Java class source file in any project path.

```
<strong>roo</strong> class [--class] [--abstract --extends --implements --path  
--permitReservedWords --rooAnnotations --force]
```

- *Mandatory:*

--class

The name of the class to create. If you consider it necessary, you can also specify the package (base package can be specified with ~). Ex.: **--class ~.domain.MyClass**. You can specify module as well, if necessary. Ex.: **--class model:~.domain.MyClass**. When working with a multi-module project, if module is not specified the class will be created in the module which has the focus.

- *Optional:*

--abstract

Whether the generated class should be marked as abstract.

Default if option present: **true**; default if option not present: **false**.

--extends

The superclass fully qualified name.

Default if option not present: **java.lang.Object**.

--implements

The interface to implement.

--path

Source directory to create the class in.

Default: *[FOCUSED-MODULE]/src/main/java*

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: **true**; default if option not present: **false**.

--rooAnnotations

Whether the generated class should have common Roo annotations (**@RooToString**, **@RooEquals** and **@RooSerializable**).

Default if option present: **true**; default if option not present: **false**.

--force

Force command execution.

Default if option present: **true**; default if option not present: **false**.

constructor

Creates a class constructor.

```
<strong>roo</strong> constructor [--class --fields]
```

- *Optional:*

--class

The name of the class to receive this constructor. If you consider it necessary, you can also

specify the package (base package can be specified with ~). Ex.: `--class ~.domain.MyEntity`. You can specify module as well, if necessary. Ex.: `--class model:~.domain.MyEntity`. When working with a multi-module project, if module is not specified, it is assumed that the class is in the module that has set the focus.

Default if option not present: the class focused by Roo shell.

--fields

The fields to include in the constructor. Multiple field names must be a double-quoted list separated by spaces.

dto

Creates a new DTO (Data Transfer Object) class in the directory `src/main/java` of the selected project module (if any) with `@RooDTO` annotation.

```
<strong>roo</strong> dto [--class] [--entityFormatExpression --entityFormatMessage  
--immutable --serializable --utilityMethods --force]
```

- *Mandatory:*

--class

The name of the DTO class to create. If you consider it necessary, you can also specify the package (base package can be specified with ~). Ex.: `--class ~.domain.MyDto`. You can specify module as well, if needed. Ex.: `--class model:~.domain.MyDto`. When working with a multi-module project, if module is not specified the class will be created in the module which has the focus.

- *Optional:*

--entityFormatExpression

The SpEL expression used to format the entity when showing it in presentation layer e.g. `{#fieldA} {#fieldB}`. It adds the `value` attribute to `io.springlets.format.EntityFormat` annotation.

--entityFormatMessage

The message key used to obtain a localized SpEL expression to format the entity when showing it in presentation layer. It adds the `message` attribute to `io.springlets.format.EntityFormat` annotation and creates a message in all message bundles with the provided key. Message value should be modified by developer. This kind of format has more priority than 'expression' format added with `--entityFormatExpression`.

--immutable

Whether the DTO should be immutable.

Default if option present: **true**; default if option not present: **false**.

--serializable

Whether the DTO should implement **java.io.Serializable**.

Default if option present: **true**; default if option not present: **false**.

--utilityMethods

Whether the DTO should implement **toString()**, **hashCode()** and **equals()** methods.

Default if option present: **true**; default if option not present: **false**.

--force

Force command execution.

Default if option present: **true**; default if option not present: **false**.

email receiver setup

Installs a Spring JavaMailReceiver in your project.

```
<strong>roo</strong> email receiver setup (--module) [--jndiName | --host --port  
--protocol --username --password --starttls] [--profile --service]
```

- *Conditional:*

--module

The application module where to install the mail configuration.

This option is mandatory if the focus is not set in an 'application' module and there are more than one 'application' modules, that is, a module containing an **@SpringBootApplication** class.

This option is available only if there are more than one application module and none of them is focused.

Default if option not present: the unique 'application' module, or focused 'application' module.

- *Optional:*

--host

The host server.

This option is not available if **--jndiName** has already been specified.

--jndiName

The jndi name where the mail configuration has been defined.

This option is not available if any of `--host`, `--port`, `--protocol`, `--username`, `--password` or `--starttls` has been specified before.

--password

The mail account password.

This option is not available if `--jndiName` has already been specified.

--port

The port used by mail server.

This option is not available if `--jndiName` has already been specified.

--profile

The profile where the properties will be set.

--protocol

The protocol used by mail server.

This option is not available if `--jndiName` has already been specified.

--service

The service where include an instance of MailReceiverService, which is a service that have methods to receive emails.

--starttls

If true, enables the use of the STARTTLS command.

This option is not available if `--jndiName` has already been specified.

--username

The mail account username.

This option is not available if `--jndiName` has already been specified.

email sender setup

Installs a Spring JavaMailSender in your project.

```
<strong>roo</strong> email sender setup (--module) [--jndiName | --host --port  
--protocol --username --password --starttls] [--profile --service]
```

- *Conditional:*

--module

The application module where to install the mail configuration.

This option is mandatory if the focus is not set in an 'application' module and there are more than one 'application' modules, that is, a module containing an `@SpringBootApplication` class.

This option is available only if there are more than one application module and none of them is focused.

Default if option not present: the unique 'application' module, or focused 'application' module.

- *Optional:*

--host

The host server.

This option is not available if `--jndiName` has already been specified.

--jndiName

The jndi name where the mail configuration has been defined.

This option is not available if any of `--host`, `--port`, `--protocol`, `--username`, `--password` or `--starttls` has been specified before.

--password

The mail account password.

This option is not available if `--jndiName` has already been specified.

--port

The port used by mail server.

This option is not available if `--jndiName` has already been specified.

--profile

The profile where the properties will be set.

--protocol

The protocol used by mail server.

This option is not available if `--jndiName` has already been specified.

--service

The service where include an instance of `JavaMailSender`, which is a service that have methods to receive emails.

--starttls

If true, enables the use of the STARTTLS command.

This option is not available if `--jndiName` has already been specified.

--username

The mail account username.

This option is not available if `--jndiName` has already been specified.

embeddable

Creates a new Java class source file with the JPA `@Embeddable` annotation in the directory `src/main/java` of the selected project module (if any).

```
<strong>roo</strong> embeddable [--class] [--permitReservedWords --serializable]
```

- *Mandatory:*

--class

The name of the embeddable class to create. If you consider it necessary, you can also specify the package (base package can be specified with `~`). Ex.: `--class ~.domain.MyEmbeddableClass`. You can specify module as well, if necessary. Ex.: `--class model:~.domain.MyEmbeddableClass`. When working with a multi-module project, if module is not specified the class will be created in the module which has the focus.

- *Optional:*

--serializable

Whether the generated class should implement `java.io.Serializable`.

Default if option present: `true`; default if option not present: `false`.

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: `true`; default if option not present: `false`.

entity jpa

Creates a new JPA persistent entity in the directory `src/main/java` of the selected project module (if any) with `@RooEntity` annotation.

```
<strong>roo</strong> entity jpa {--class} (--identifierColumn --identifierStrategy
--table --sequenceName --versionField --versionColumn --versionType) [--
entityFormatExpression --entityFormatMessage --abstract --catalog --entityName --extends
--identifierField --identifierType --implements --inheritanceType --mappedSuperclass
--permitReservedWords --plural --readOnly --schema --serializable --force]
```

- *Mandatory:*

--class

The name of the entity to create. If you consider it necessary, you can also specify the package (base package can be specified with ~). Ex.: `--class ~.domain.MyEntity`. You can specify module as well, if necessary. Ex.: `--class model:~.domain.MyEntity`. When working with a multi-module project, if module is not specified the entity will be created in the module which has the focus.

- *Conditional:*

All the following parameters are mandatory if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's value is `true`.

--identifierColumn

The JPA identifier field column to use for this entity.

--identifierStrategy

The generation value strategy to be used.

Default if option present: `AUTO`.

--table

The JPA table name to use for this entity.

--sequenceName

The name of the sequence for incrementing sequence-driven primary keys.

--versionField

The JPA version field name to use for this entity.

--versionColumn

The JPA version field column to use for this entity.

This option is available only when `--versionField` has been specified.

--versionType

The data type that will be used for the JPA version field.

This option is available only when `--versionField` has been specified.

- *Optional:*

--entityFormatExpression

The SpEL expression used to format the entity when showing it in presentation layer e.g. `{#fieldA} {#fieldB}`. It adds the `value` attribute to `io.springlets.format.EntityFormat` annotation.

--entityFormatMessage

The message key used to obtain a localized SpEL expression to format the entity when showing it in presentation layer. It adds the `message` attribute to `io.springlets.format.EntityFormat` annotation and creates a message in all message bundles with the provided key. Message value should be modified by developer. This kind of format has more priority than 'expression' format added with `--entityFormatExpression`.

--extends

The fully qualified name of the superclass.

Default if option not present: `java.lang.Object`.

--implements

The fully qualified name of the interface to implement.

--abstract

Whether the generated class should be marked as abstract.

Default if option present: `true`; default if option not present: `false`.

--schema

The JPA table schema name to use for this entity.

--catalog

The JPA table catalog name to use for this entity.

--identifierField

The JPA identifier field name to use for this entity.

--identifierType

The data type that will be used for the JPA identifier field.

Default: `java.lang.Long`.

--inheritanceType

The JPA `@Inheritance` value (apply to base class).

--mappedSuperclass

Apply `@MappedSuperclass` for this entity.

Default if option present: `true`; default if option not present: `false`.

--serializable

Whether the generated class should implement `java.io.Serializable`.

Default if option present: `true`; default if option not present: `false`.

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: `true`; default if option not present: `false`.

--entityName

The name used to refer to the entity in queries.

--readOnly

Whether the generated entity should be used for read operations only.

Default if option present: `true`; default if option not present: `false`.

--plural

Specify the plural of this new entity. If not provided, a calculated plural will be used by default.

--force

Force command execution.

Default if option present: `true`; default if option not present: `false`.

entity projection

Creates new projection classes from entities in the directory `src/main/java` of the selected project module (if any) annotated with `@RooEntityProjection`. Transient, static and entity collection fields are not valid for projections.

```
<strong>roo</strong> entity projection (--all [--suffix] | --class --entity --fields [--entityFormatExpression --entityFormatMessage]) [--force]
```

- *Conditional:*

--all

Create one projection class for each entity in the project.

This option is mandatory if `--class` is not specified. Otherwise, using `--class` will cause the parameter `--all` won't be available.

--class

The name of the projection class to create. If you consider it necessary, you can also specify the package (base package can be specified with `~`). Ex.: `--class ~.domain.MyProjection`. You can specify module as well, if necessary. Ex.: `--class model:~.domain.MyProjection`. When working with a multi-module project, if module is not specified the projection will be created in the module which has the focus.

This option is mandatory if `--all` is not specified. Otherwise, using `--all` will cause the parameter `--class` won't be available.

--entity

Name of the entity which can be used to create the Projection from.

This option is mandatory if `--class` is specified. Otherwise, not specifying `--class` will cause the parameter `--entity` won't be available.

--fields

Comma separated list of entity fields to be included into the Projection.

Possible values are: non-static, nor transient, nor entity collection fields from main entity or its related entities (only for one-to-one or many-to-one relations).

This option is mandatory if `--class` is specified. Otherwise, not specifying `--class` will cause the parameter `--fields` won't be available.

• Optional:

--entityFormatExpression

The SpEL expression used to format the entity when showing it in presentation layer e.g. `{#fieldA} {#fieldB}`. It adds the `value` attribute to `io.springlets.format.EntityFormat` annotation.

This option is available only if `--entity` has been specified.

--entityFormatMessage

The message key used to obtain a localized SpEL expression to format the entity when showing it in presentation layer. It adds the `message` attribute to `io.springlets.format.EntityFormat` annotation and creates a message in all message bundles with the provided key. Message value should be modified by developer. This kind of format has more priority that 'expression' format added with `--entityFormatExpression`.

This option is available only if `--entity` has been specified.

--suffix

Suffix added to each Projection class name, built from each associated entity name.

This option is only available if **--all** has been already specified.

Default if option not present: 'Projection'.

--force

Force command execution Default if option present: **true**; default if option not present: **false**.

enum constant

Inserts a new enum constant into an enum class.

```
<strong>roo</strong> enum constant [--name] [--class --permitReservedWords]
```

- *Mandatory:*

--name

The name of the constant. It will converted to upper case automatically.

- *Optional:*

--class

The name of the enum class to receive this constant. When working on a mono module project, simply specify the name of the class in which the new constant will be included. If you consider it necessary, you can also specify the package. Ex.: **--class ~.domain.MyEnumClass** (where ~ is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: **--class model:~.domain.MyEnumClass**. If the module is not specified, it is assumed that the class is in the module which has the focus.

Default if option not present: the class focused by Roo shell.

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: **true**; default if option not present: **false**.

enum type

Creates a new Java enum source file in any project path

```
<strong>roo</strong> enum type [--class] [--path --permitReservedWords --force]
```

- *Mandatory:*

--class

The name of the enum class to create. If you consider it necessary, you can also specify the package (base package can be specified with ~). Ex.: `--class ~.domain.MyEnumClass`. You can specify module as well, if necessary. Ex.: `--class model:~.domain.MyEnumClass`. When working with a multi-module project, if module is not specified the projection will be created in the module which has the focus.

- *Optional:*

--path

Source directory where create the enum.

Default: `[FOCUSED-MODULE]/src/main/java`

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: `true`; default if option not present: `false`.

--force

Force command execution.

Default if option present: `true`; default if option not present: `false`.

equals

Adds `equals()` and `hashCode()` methods to a class.

```
<strong>roo</strong> equals [--class --appendSuper --excludeFields]
```

- *Optional:*

--class

The name of the class to generate `equals()` and `hashCode()` methods. When working on a mono module project, simply specify the name of the class in which the methods will be included. If you consider it necessary, you can also specify the package. Ex.: `--class ~.domain.MyClass` (where ~ is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class model:~.domain.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

Default if option not present: the class focused by Roo shell.

--appendSuper

Whether to call the super class `equals()` and `hashCode()` methods. This param has no effect when used against JPA entities.

Default if option present: `true`; default if option not present: `false`.

--excludeFields

The fields to exclude in the `equals()` and `hashCode()` methods. Multiple field names must be a double-quoted list separated by spaces.

exit

Waits until all metadata and files are refreshed and updated, then exits the shell. You can also use `quit` command.

```
<strong>roo</strong> exit
```

This command does not accept any options.

field boolean

Adds a private boolean field to an existing Java source file.

```
<strong>roo</strong> field boolean {--fieldName} (--class --column [--transient]) [--assertFalse | --assertTrue] [--notNull --value --comment --primitive --permitReservedWords --force]
```

- *Mandatory:*

--fieldName

The name of the field to add.

- *Conditional:*

--class

The name of the class to generate the field. When working on a mono module project, simply specify the name of the class in which the field will be included. If you consider it necessary, you can also specify the package. Ex.: `--class ~.domain.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class model:~.domain.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is mandatory for this command when the focus is not set to one class.

Default if option not present: the class focused by Roo shell.

--column

The JPA @Column name.

This option is mandatory if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available for JPA entities and embeddable classes.

--transient

Indicates to mark the field as transient, adding JPA `javax.persistence.Transient` annotation. This marks the field as not persistent.

This option is only available for JPA entities and embeddable classes.

Default if option present: `true`. Default if option not present: `false`.

• *Optional:*

--assertFalse

Whether the value of this field must be false. Adds `javax.validation.constraints.AssertFalse` annotation to the field.

This option is not available if `--assertTrue` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--assertTrue

Whether the value of this field must be true. Adds `javax.validation.constraints.AssertTrue` annotation to the field.

This option is not available if `--assertFalse` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--notNull

Whether this value cannot be null. Adds `javax.validation.constraints.NotNull` annotation to the field.

Default if option present: `true`; default if option not present: `false`.

--value

Inserts an optional Spring `org.springframework.beans.factory.annotation.Value` annotation with the given content, typically used for expression-driven dependency injection.

--comment

An optional comment for JavaDocs.

--primitive

Indicates to use the primitive type.

Default if option present: **true**; default if option not present: **false**.

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: **true**; default if option not present: **false**.

--force

Force command execution.

Default if option present: **true**; default if option not present: **false**.

field date

Adds a private date field to an existing Java source file.

```
<strong>roo</strong> field date {--fieldName --type} (--class --column [--persistenceType --transient]) [--notNull | --nullRequired] [--future | --past] [--dateTimeFormatPattern | --dateFormat --timeFormat] [--comment --value --permitReservedWords --force]
```

- *Mandatory:*

--fieldName

The name of the field to add.

--type

The Java date type of the field. Its value can be **java.util.Date** or **java.util.Calendar**.

- *Conditional:*

--class

The name of the class to generate the field. When working on a mono module project, simply specify the name of the class in which the field will be included. If you consider it necessary, you can also specify the package. Ex.: **--class ~.domain.MyClass** (where **~** is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: **--class model:~.domain.MyClass**. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is mandatory for this command when the focus is not set to one class.

Default if option not present: the class focused by Roo shell.

--column

The JPA @Column name.

This option is mandatory if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available for JPA entities and embeddable classes.

--persistenceType

The type of persistent storage to be used. It adds a `javax.persistence.TemporalType` to a `javax.persistence.Temporal` annotation into the field.

This option is only available for JPA entities and embeddable classes.

Default if option not present: `TemporalType.TIMESTAMP`

--transient

Indicates to mark the field as transient, adding JPA `javax.persistence.Transient` annotation. This marks the field as not persistent.

This option is only available for JPA entities and embeddable classes.

Default if option present: `true`. Default if option not present: `false`.

• *Optional:*

--notNull

Whether this value cannot be null. Adds `javax.validation.constraints.NotNull` annotation to the field.

This option is not available if `--nullRequired` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--nullRequired

Whether this value must be null. Adds `javax.validation.constraints.Null` annotation to the field.

This option is not available if `--notNull` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--future

Whether this value must be in the future. Adds `field.java.validation.constraints.Future` annotation to the field.

This option is not available if `--past` option has already been specified.

Default if option present: `true`; default if option not present: `false`.

--past

Whether this value must be in the past. Adds `field.java.validation.constraints.Past` annotation to the field.

This option is not available if `--future` option has already been specified.

Default if option present: `true`; default if option not present: `false`.

--dateFormat

Indicates the style of the date format, adding `style` attribute to `org.springframework.format.annotation.DateTimeFormat` annotation into the field, with date style (first character of the code).

Possible values are: `MEDIUM` (style='M-'), `NONE` (style='--') and `SHORT` (style='S-').

This option is not available if `--dateTimeFormatPattern` has already been specified.

Default: `MEDIUM`.

--timeFormat

Indicates the style of the time format, adding `style` attribute to `org.springframework.format.annotation.DateTimeFormat` annotation into the field, with time style (second character of the code).

Possible values are: `MEDIUM` (style='-M'), `NONE` (style='--') and `SHORT` (style='-S').

This option is not available if `--dateTimeFormatPattern` has already been specified.

Default: `NONE`.

--dateTimeFormatPattern

Indicates a 'custom' DateTime format pattern such as yyyy-MM-dd hh:mm:ss, adding `pattern` attribute to `org.springframework.format.annotation.DateTimeFormat` annotation into the field, with the provided value.

This option is not available if `--timeFormat` or `--dateFormat` have already been specified.

--comment

An optional comment for JavaDocs.

--value

Inserts an optional Spring `org.springframework.beans.factory.annotation.Value` annotation with the given content, typically used for expression-driven dependency injection.

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: `true`; default if option not present: `false`.

--force

Force command execution.

Default if option present: `true`; default if option not present: `false`.

field embedded

Adds a private `@Embedded` field to an existing Java source file. This command is only available for entities annotated with `@RooJpaEntity`. Therefore, you should focus the desired entity in the Roo Shell to make this command available.

```
<strong>roo</strong> field embedded [--fieldName --type] (--class) [--  
permitReservedWords --force]
```

- *Mandatory:*

--fieldName

The name of the field to add.

--type

The Java type of an embeddable class, annotated with `@Embeddable`.

Possible values are: any class in the project annotated with `@Embeddable`.

- *Conditional:*

--class

The name of the class to generate the field. When working on a mono module project, simply specify the name of the class in which the field will be included. If you consider it necessary, you can also specify the package. Ex.: `--class ~.domain.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class model:~.domain.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is mandatory for this command when the focus is not set to one class.

Default if option not present: the class focused by Roo Shell.

- *Optional:*

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: **true**; default if option not present: **false**.

--force

Force command execution.

Default if option present: **true**; default if option not present: **false**.

field enum

Adds a private enum field to an existing Java source file. The field type must be a Java enum type.

```
<strong>roo</strong> field enum {--fieldName --type} (--class --column [--transient  
--enumType]) [--notNull | --nullRequired] [--comment --permitReservedWords --force]
```

- *Mandatory:*

--fieldName

The name of the field to add.

--type

The Java type of the field. It must be a Java enum type.

Possible values are: any enumerated class in the user's project.

- *Conditional:*

--class

The name of the class to generate the field. When working on a mono module project, simply specify the name of the class in which the field will be included. If you consider it necessary, you can also specify the package. Ex.: **--class ~.domain.MyClass** (where ~ is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: **--class model:~.domain.MyClass**. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is mandatory for this command when the focus is not set to one class.

Default if option not present: the class focused by Roo Shell.

--column

The JPA `@Column` name.

This option is mandatory if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available for JPA entities and embeddable classes.

--transient

Indicates to mark the field as transient, adding JPA `javax.persistence.Transient` annotation. This marks the field as not persistent.

This option is only available for JPA entities and embeddable classes.

Default if option present: `true`; default if option not present: `false`.

--enumType

Defines how the enumerated field should be persisted at a JPA level. Adds the `javax.persistence.Enumerated` annotation to the field, with `javax.persistence.EnumType` attribute.

Possible values are: `ORDINAL` (persists as an integer) and `STRING` (persists as a String). If this option is not specified, the `Enumerated` annotation will be added without the `EnumType` attribute, using its default value (`ORDINAL`).

This option is only available for JPA entities and embeddable classes.

- *Optional:*

--notNull

Whether this value cannot be null. Adds `javax.validation.constraints.NotNull` annotation to the field.

This option is not available if `--nullRequired` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--nullRequired

Whether this value must be null. Adds `javax.validation.constraints.Null` annotation to the field.

This option is not available if `--notNull` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--comment

An optional comment for JavaDocs.

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: **true**; default if option not present: **false**.

--force

Force command execution.

Default if option present: **true**; default if option not present: **false**.

field file

Adds a byte array field for storing uploaded file contents.

```
<strong>roo</strong> field file [--fieldName --contentType] (--class --column) [--autoUpload --notNull --permitReservedWords --force]
```

- *Mandatory:*

--fieldName

The name of the file upload field to add.

--contentType

The content type of the file.

Possible values are: **CSS**, **CSV**, **DOC**, **GIF**, **HTML**, **JAVASCRIPT**, **JPG**, **JSON**, **MP3**, **MP4**, **MPEG**, **PDF**, **PNG**, **TXT**, **XLS**, **XML** and **ZIP**.

- *Conditional:*

--class

The name of the class to generate the field. When working on a mono module project, simply specify the name of the class in which the field will be included. If you consider it necessary, you can also specify the package. Ex.: **--class ~.domain.MyClass** (where **~** is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: **--class model:~.domain.MyClass**. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is mandatory for this command when the focus is not set to one class.

Default if option not present: the class focused by Roo Shell.

--column

The JPA **@Column** name.

This option is mandatory if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available for JPA entities and embeddable classes.

- *Optional:*

- **--autoUpload**

- Whether the file is uploaded automatically when selected.

- Default if option present: `true`; default if option not present: `false`.

- **--notNull**

- Whether this value cannot be null. Adds `javax.validation.constraints.NotNull` annotation to the field.

- Default if option present: `true`; default if option not present: `false`.

- **--permitReservedWords**

- Indicates whether reserved words are ignored by Roo.

- Default if option present: `true`; default if option not present: `false`.

- **--force**

- Force command execution.

- Default if option present: `true`; default if option not present: `false`.

field list

Adds a private `List` field to an existing Java source file, representing (always) a bidirectional relation with other entity. Therefore, this command will also add a field on the other side of the relation (the owner side, with `mappedBy` attribute), which will be a `List` field for 'many-to-many' relations, or a `not Collection` field for a 'one-to-many' relation. All added fields will have the needed JPA annotations to properly manage bidirectional relations.

This command is only available for entities annotated with `@RooJpaEntity` (Roo JPA entities). Therefore, you should focus the desired entity in the Roo Shell to make this command available.

```
<strong>roo</strong> field list [--fieldName --type] (--class) (--joinColumnName
--referencedColumnName | --joinTable --joinColumns --referencedColumns
--inverseJoinColumns --inverseReferencedColumns) [--notNull | --nullRequired] [--mappedBy
--cardinality --fetch --aggregation --comment --entityFormatExpression
--entityFormatMessage --orphanRemoval --sizeMin --sizeMax --permitReservedWords --force]
```

- *Mandatory:*

--fieldName

The name of the field to add.

--type

The entity related to this one, which will be contained within the `List`.

Possible values are: any of the entities in the project.

- *Conditional:*

--class

The name of the class to generate the field. When working on a mono module project, simply specify the name of the class in which the field will be included. If you consider it necessary, you can also specify the package. Ex.: `--class ~.domain.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class model:~.domain.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is mandatory for this command when the focus is not set to one class.

Default if option not present: the class focused by Roo Shell.

--joinColumnName

The JPA `@JoinColumn name` attribute. When this option is set, cardinality will be set as `ONE_TO_MANY`.

This option is mandatory for 'ONE_TO_MANY' relationships without join table, if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

--referencedColumnName

The JPA `@JoinColumn referencedColumnName` attribute.

This option is only available when `--joinColumnName` option is set.

--joinTable

Join table name. Most usually used in `@ManyToMany` relations.

This option is mandatory for this command if `--cardinality` is set to `MANY_TO_MANY` and `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

--joinColumns

Comma separated list of join table's foreign key columns which references the table of the related entity (the owner entity in bidirectional relations).

This option is mandatory if `--joinTable` option has been specified and if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available when `--joinTable` option is set.

--referencedColumns

Comma separated list of foreign key referenced columns in the primary table of the related entity (the owner entity in bidirectional relations).

This option is mandatory if `--joinTable` option has been specified and if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available when `--joinTable` option is set.

--inverseJoinColumns

Comma separated list of join table's foreign key columns which references the table of the entity that does not own the relation (current entity).

This option is mandatory if `--joinTable` option has been specified and if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available when `--joinTable` option is set.

--inverseReferencedColumns

Comma separated list of foreign key referenced columns in the primary table of the entity that does not own the relation (current entity).

This option is mandatory if `--joinTable` option has been specified and if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available when `--joinTable` option is set.

- *Optional:*

--mappedBy

The field name on the referenced type which owns the relationship, which will be also created due to bidirectional relation. If the field already exists in the related entity, command won't be executed.

Default if not present: current entity name in lower camel case.

--cardinality

The relationship cardinality at a JPA level.

Default: `ONE_TO_MANY`.

--fetch

The fetch semantics at a JPA level. It adds the provided value to `fetch` attribute of JPA `@OneToMany`, `@ManyToMany` and `@ManyToOne`.

Possible values are `LAZY`` and ``EAGER`.

Default if option not present: `LAZY`.

--notNull

Whether this value cannot be null. Adds `javax.validation.constraints.NotNull` annotation to the field.

This option is not available if `--nullRequired` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--nullRequired

Whether this value must be null. Adds `javax.validation.constraints.Null` annotation to the field.

This option is not available if `--notNull` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--aggregation

Whether the relationship type is 'aggregation' or 'composition'. An aggregation relation means that children entities aren't dependent from parent entity (current entity) and they can exist without parent entity. In the other hand, in a composition relation the parent entity of the relationship also owns the life cycle of related entities. The parent entity is responsible for the creation and destruction of children entities, these being linked to a single parent entity. A child entity cannot be in two different composition relationships.

Default: `true`.

--entityFormatExpression

The SpEL expression used to format the related entity when showing it in presentation layer e.g. `{#fieldA} {#fieldB}`. It adds the `value` attribute to `io.springlets.format.EntityFormat` annotation.

--entityFormatMessage

The message key used to obtain a localized SpEL expression to format the related entity when showing it in presentation layer. It adds the `message` attribute to `io.springlets.format.EntityFormat` annotation and creates a message in all message bundles with the provided key. Message value should be modified by developer.

--orphanRemoval

Indicates whether to apply the remove operation to entities that have been removed from the relationship and to cascade the remove operation to those entities. If this relation represents a 'composition' relation and this option is not present, `--orphanRemoval` value will be `true`.

Default if option present: `true`.

--sizeMin

The minimum number of elements in the collection. This option adds or updates `javax.validation.constraints.Size` with the provided value as `min` attribute value.

--sizeMax

The maximum number of elements in the collection. This option adds or updates `javax.validation.constraints.Size` with the provided value as `max` attribute value.

--comment

An optional comment for JavaDocs.

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: `true`; default if option not present: `false`.

--force

Force command execution.

Default if option present: `true`; default if option not present: `false`.

field number

Adds a private numeric field to an existing Java source file. User can choose the field type between a wide range of numeric types.

```
<strong>roo</strong> field number {--fieldName --type} (--class --column [--unique
--transient]) [--nullRequired | --notNull --primitive] [--decimalMin --decimalMax
--digitsInteger --digitsFraction --min --max --comment --value --permitReservedWords
--force]
```

- *Mandatory:*

--fieldName

The name of the field to add.

--type

The Java type of the field. Only numeric types allowed.

Possible values are: `java.math.BigDecimal`, `java.math.BigInteger`, `byte`, `java.lang.Byte`, `double`, `java.lang.Double`, `float`, `java.lang.Float`, `int`, `java.lang.Integer`, `long`, `java.lang.Long`, `java.lang.Number`, `short` and `java.lang.Short`.

- *Conditional:*

- class**

The name of the class to generate the field. When working on a mono module project, simply specify the name of the class in which the field will be included. If you consider it necessary, you can also specify the package. Ex.: `--class ~.domain.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class model:~.domain.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is mandatory for this command when the focus is not set to one class.

Default if option not present: the class focused by Roo Shell.

- column**

The JPA `@Column` name.

This option is mandatory if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available for JPA entities and embeddable classes.

- unique**

Indicates whether to mark the field with a unique constraint.

This option is only available for JPA entities and embeddable classes.

Default if option present: `true`; default if option not present: `false`.

- transient**

Indicates to mark the field as transient, adding JPA `javax.persistence.Transient` annotation. This marks the field as not persistent.

This option is only available for JPA entities and embeddable classes.

Default if option present: `true`. Default if option not present: `false`.

- *Optional:*

- notNull**

Whether this value cannot be null. Adds `javax.validation.constraints.NotNull`

annotation to the field.

This option is not available if `--nullRequired` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--nullRequired

Whether this value must be null. Adds `javax.validation.constraints.Null` annotation to the field.

This option is not available if `--notNull` or `--primitive` option have already been specified with value `true` or without value.

Default if option present: `true`; default if option not present: `false`.

--decimalMin

The BigDecimal string-based representation of the minimum value. It adds to the field `javax.validation.constraints.DecimalMin` annotation with provided value.

--decimalMax

The BigDecimal string based representation of the maximum value. It adds to the field `javax.validation.constraints.DecimalMax` annotation with provided value.

--digitsInteger

Maximum number of integral digits accepted for this number. It creates or updates field `javax.validation.constraints.Digits` annotation, adding `integer` attribute with the provided value.

--digitsFraction

Maximum number of fractional digits accepted for this number. It creates or updates field `javax.validation.constraints.Digits` annotation, adding `fraction` attribute with the provided value.

--min

The minimum value of the numeric field. It adds `javax.validation.constraints.Min` with provided value to the field.

--max

The maximum value of the numeric field. It adds `javax.validation.constraints.Max` with provided value to the field.

--comment

An optional comment for JavaDocs.

--value

Inserts an optional Spring `org.springframework.beans.factory.annotation.Value` annotation with the given content, typically used for expression-driven dependency injection.

--primitive

Indicates to use a primitive type if possible.

Default if option present: `true`; default if option not present: `false`.

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: `true`; default if option not present: `false`.

--force

Force command execution.

Default if option present: `true`; default if option not present: `false`.

field other

Inserts a private field into the specified file. User can choose a custom type for the field by specifying its fully qualified name.

```
<strong>roo></strong> field other {--fieldName --type} (--class --column [--transient])  
[--notNull | --nullRequired] [--comment --value --permitReservedWords --force]
```

- *Mandatory:*

--fieldName

The name of the field.

--type

The Java type of this field.

- *Conditional:*

--class

The name of the class to generate the field. When working on a mono module project, simply specify the name of the class in which the field will be included. If you consider it necessary, you can also specify the package. Ex.: `--class ~.domain.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class model:~.domain.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is mandatory for this command when the focus is not set to one class.

Default if option not present: the class focused by Roo Shell.

--column

The JPA `@Column` name.

This option is mandatory if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available for JPA entities and embeddable classes.

--transient

Indicates to mark the field as transient, adding JPA `javax.persistence.Transient` annotation. This marks the field as not persistent.

This option is only available for JPA entities and embeddable classes.

Default if option present: `true`. Default if option not present: `false`

• *Optional:*

--notNull

Whether this value cannot be null. Adds `javax.validation.constraints.NotNull` annotation to the field.

This option is not available if `--nullRequired` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--nullRequired

Whether this value must be null. Adds `javax.validation.constraints.Null` annotation to the field.

This option is not available if `--notNull` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--comment

An optional comment for JavaDocs.

--value

Inserts an optional Spring `org.springframework.beans.factory.annotation.Value` annotation with the given content, typically used for expression-driven dependency injection.

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: **true**; default if option not present: **false**.

--force

Force command execution.

Default if option present: **true**; default if option not present: **false**.

field reference

Adds a private reference field, representing (always) a bidirectional 'one-to-one' relation, to an existing Java source file. Therefore, this command will add as well a 'one-to-one' field on the other side of the relation.

This command is only available for entities annotated with `@RooJpaEntity`, so you should focus the desired entity in the Roo Shell to make this command available.

```
<strong>roo</strong> field reference [--fieldName --type] (--class --joinColumnName  
--referencedColumnName [--fetch --mappedBy]) [--notNull | --nullRequired] [--aggregation  
--entityFormatExpression --entityFormatMessage --orphanRemoval --comment  
--permitReservedWords --force]
```

- *Mandatory:*

--fieldName

The name of the field to add.

--type

The Java type of the entity to reference.

Possible values are: any of the entities in the project.

- *Conditional:*

--class

The name of the class to generate the field. When working on a mono module project, simply specify the name of the class in which the field will be included. If you consider it necessary, you can also specify the package. Ex.: `--class ~.domain.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class model:~.domain.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is mandatory for this command when the focus is not set to one class.

Default if option not present: the class focused by Roo Shell.

--joinColumnName

The JPA `@JoinColumn name` attribute.

This option is mandatory if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available for JPA entities.

--referencedColumnName

The JPA `@JoinColumn referencedColumnName` attribute.

This option is only available for JPA entities.

--fetch

The fetch semantics at a JPA level. It adds the provided value to `fetch` attribute of JPA `@OneToOne`. If this option is not provided, default fetch type will be `LAZY`.

Possible values are `LAZY`` and ``EAGER`.

This option is only available for JPA entities and embeddable classes.

--mappedBy

The field name on the referenced type which owns the relationship, which will be also created due to bidirectional relation. If not specified, it will take the lower camel case of the current entity (focused entity or specified in `--class` option). If the field already exists in the related entity, command won't be executed.

This option is only available for JPA entities.

Default if not present: current entity name in lower camel case.

- *Optional:*

--notNull

Whether this value cannot be null. Adds `javax.validation.constraints.NotNull` annotation to the field.

This option is not available if `--nullRequired` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--nullRequired

Whether this value must be null. Adds `javax.validation.constraints.Null` annotation to the field.

This option is not available if `--notNull` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--aggregation

Whether the relationship type is 'aggregation' or 'composition'. An aggregation relation means that children entities aren't dependent from parent entity (current entity) and they can exist without parent entity. In the other hand, in a composition relation the parent entity of the relationship also owns the life cycle of related entities. The parent entity is responsible for the creation and destruction of children entities, these being linked to a single parent entity. A child entity cannot be in two different composition relationships.

Default: `true`.

--entityFormatExpression

The SpEL expression used to format the related entity when showing it in presentation layer e.g. `{#fieldA} {#fieldB}`. It adds the `value` attribute to `io.springlets.format.EntityFormat` annotation.

--entityFormatMessage

The message key used to obtain a localized SpEL expression to format the related entity when showing it in presentation layer. It adds the `message` attribute to `io.springlets.format.EntityFormat` annotation and creates a message in all message bundles with the provided key. Message value should be modified by developer.

--orphanRemoval

Indicates whether to apply the remove operation to entities that have been removed from the relationship and to cascade the remove operation to those entities. If this relation represents a 'composition' relation and this option is not present, `--orphanRemoval` value will be `true`.

Default if option present: `true`.

--comment

An optional comment for JavaDocs.

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: `true`; default if option not present: `false`.

--force

Force command execution.

Default if option present: `true`; default if option not present: `false`.

field set

Adds a private **Set** field to an existing Java source file, representing (always) a bidirectional relation with other entity. Therefore, this command will also add a field on the other side of the relation (the owner side, with **mappedBy** attribute), which will be a **Set** field for 'many-to-many' relations, or a **not Collection** field for a 'one-to-many' relation. All added fields will have the needed JPA annotations to properly manage bidirectional relations.

This command is only available for entities annotated with **@RooJpaEntity** (Roo JPA entities). Therefore, you should focus the desired entity in the Roo Shell to make this command available.

```
<strong>roo</strong> field set {--fieldName --type} (--class) (--joinColumnName
--referencedColumnName | --joinTable --joinColumns --referencedColumns
--inverseJoinColumns --inverseReferencedColumns) [--notNull | --nullRequired] [--mappedBy
--cardinality --fetch --aggregation --comment --entityFormatExpression
--entityFormatMessage --orphanRemoval --sizeMin --sizeMax --permitReservedWords --force]
```

- *Mandatory:*

- fieldName**

- The name of the field to add.

- type**

- The entity related to this one, which will be contained within the **List**.

- Possible values are: any of the entities in the project.

- *Conditional:*

- class**

- The name of the class to generate the field. When working on a mono module project, simply specify the name of the class in which the field will be included. If you consider it necessary, you can also specify the package. Ex.: **--class ~.domain.MyClass** (where **~** is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: **--class model:~.domain.MyClass**. If the module is not specified, it is assumed that the class is in the module which has the focus.

- This option is mandatory for this command when the focus is not set to one class.

- Default if option not present: the class focused by Roo Shell.

- joinColumnName**

- The JPA **@JoinColumn name** attribute. When this option is set, cardinality will be set as **ONE_TO_MANY**.

This option is mandatory for 'ONE_TO_MANY' relationships without join table, if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

--referencedColumnName

The JPA `@JoinColumn` `referencedColumnName` attribute.

This option is only available when `--joinColumnName` option is set.

--joinTable

Join table name. Most usually used in `@ManyToMany` relations.

This option is mandatory for this command if `--cardinality` is set to `MANY_TO_MANY` and `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

--joinColumns

Comma separated list of join table's foreign key columns which references the table of the related entity (the owner entity in bidirectional relations).

This option is mandatory if `--joinTable` option has been specified and if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available when `--joinTable` option is set.

--referencedColumns

Comma separated list of foreign key referenced columns in the primary table of the related entity (the owner entity in bidirectional relations).

This option is mandatory if `--joinTable` option has been specified and if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available when `--joinTable` option is set.

--inverseJoinColumns

Comma separated list of join table's foreign key columns which references the table of the entity that does not own the relation (current entity).

This option is mandatory if `--joinTable` option has been specified and if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available when `--joinTable` option is set.

--inverseReferencedColumns

Comma separated list of foreign key referenced columns in the primary table of the entity that does not own the relation (current entity).

This option is mandatory if `--joinTable` option has been specified and if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available when `--joinTable` option is set.

- *Optional:*

--mappedBy

The field name on the referenced type which owns the relationship, which will be also created due to bidirectional relation. If the field already exists in the related entity, command won't be executed.

Default if not present: current entity name in lower camel case.

--cardinality

The relationship cardinality at a JPA level.

Default: `ONE_TO_MANY`.

--fetch

The fetch semantics at a JPA level. It adds the provided value to `fetch` attribute of JPA `@OneToMany`, `@ManyToMany` and `@ManyToOne`.

Possible values are `LAZY`` and ``EAGER`.

Default if option not present: `LAZY`.

--aggregation

Whether the relationship type is 'aggregation' or 'composition'. An aggregation relation means that children entities aren't dependent from parent entity (current entity) and they can exist without parent entity. In the other hand, in a composition relation the parent entity of the relationship also owns the life cycle of related entities. The parent entity is responsible for the creation and destruction of children entities, these being linked to a single parent entity. A child entity cannot be in two different composition relationships.

Default: `true`.

--notNull

Whether this value cannot be null. Adds `javax.validation.constraints.NotNull` annotation to the field.

This option is not available if `--nullRequired` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--nullRequired

Whether this value must be null. Adds `javax.validation.constraints.Null` annotation to the field.

This option is not available if `--notNull` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--entityFormatExpression

The SpEL expression used to format the related entity when showing it in presentation layer e.g. `{#fieldA} {#fieldB}`. It adds the `value` attribute to `io.springlets.format.EntityFormat` annotation.

--entityFormatMessage

The message key used to obtain a localized SpEL expression to format the related entity when showing it in presentation layer. It adds the `message` attribute to `io.springlets.format.EntityFormat` annotation and creates a message in all message bundles with the provided key. Message value should be modified by developer.

--orphanRemoval

Indicates whether to apply the remove operation to entities that have been removed from the relationship and to cascade the remove operation to those entities. If this relation represents a 'composition' relation and this option is not present, `--orphanRemoval` value will be `true`.

Default if option present: `true`.

--sizeMin

The minimum number of elements in the collection. This option adds or updates `javax.validation.constraints.Size` with the provided value as `min` attribute value.

--sizeMax

The maximum number of elements in the collection. This option adds or updates `javax.validation.constraints.Size` with the provided value as `max` attribute value.

--comment

An optional comment for JavaDocs.

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: `true`; default if option not present: `false`.

--force

Force command execution.

Default if option present: `true`; default if option not present: `false`.

field string

Adds a private string field to an existing Java source file.

```
<strong>roo</strong> field string {--fieldName} (--class --column [--transient --lob  
--unique]) [--notNull | --nullRequired] [--regexp --sizeMin --sizeMax --value --comment  
--permitReservedWords --force]
```

- *Mandatory:*

--fieldName

The name of the field to add.

- *Conditional:*

--class

The name of the class to generate the field. When working on a mono module project, simply specify the name of the class in which the field will be included. If you consider it necessary, you can also specify the package. Ex.: `--class ~.domain.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class model:~.domain.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is mandatory for this command when the focus is not set to one class.

Default if option not present: the class focused by Roo Shell.

--column

The JPA `@Column` name.

This option is mandatory if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

This option is only available for JPA entities and embeddable classes.

--transient

Indicates to mark the field as transient, adding JPA `javax.persistence.Transient` annotation. This marks the field as not persistent.

This option is only available for JPA entities and embeddable classes.

Default if option present: `true`. Default if option not present: `false`

--lob

Indicates that this field is a Large Object. This option adds `javax.persistence.Lob` annotation

to the field.

This option is only available for JPA entities and embeddable classes.

Default if option present: `true`; default if option not present: `false`.

--unique

Indicates whether to mark the field with a unique constraint.

This option is only available for JPA entities and embeddable classes.

Default if option present: `true`; default if option not present: `false`.

• *Optional:*

--notNull

Whether this value cannot be null. Adds `javax.validation.constraints.NotNull` annotation to the field.

This option is not available if `--nullRequired` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--nullRequired

Whether this value must be null. Adds `javax.validation.constraints.Null` annotation to the field.

This option is not available if `--notNull` has already been specified.

Default if option present: `true`; default if option not present: `false`.

--regexp

The required regular expression pattern. This option adds `javax.validation.constraints.Pattern` with the provided value as `regexp` attribute.

--sizeMin

The minimum string length. This option adds or updates `javax.validation.constraints.Size` with the provided value as `min` attribute value.

--sizeMax

The maximum string length. This option adds or updates `javax.validation.constraints.Size` with the provided value as `max` attribute value.

--value

Inserts an optional Spring `org.springframework.beans.factory.annotation.Value` annotation with the given content, typically used for expression-driven dependency injection.

--comment

An optional comment for JavaDocs.

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: **true**; default if option not present: **false**.

--force

Force command execution.

Default if option present: **true**; default if option not present: **false**.

finder add

Installs a finder in the given target (must be an entity). This command needs an existing repository for the target entity, you can create it with **repository jpa** command. The finder will be added to targeted entity associated repository and associated service if exists or when it will be created.

```
<strong>roo</strong> finder add {--entity --name} (--formBean --returnType)
```

- *Mandatory:*

--entity

The entity for which the finders are generated. When working on a mono module project, simply specify the name of the entity. If you consider it necessary, you can also specify the package. Ex.: **--class ~.domain.MyEntity** (where ~ is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: **--class model:~.domain.MyEntity**. If the module is not specified, it is assumed that the entity is in the module which has the focus.

--name

The finder string defined as a Spring Data query. Use Spring Data JPA nomenclature.

Possible values are: any finder name following Spring Data nomenclature.

This option will not be available until **--entity** is specified.

- *Conditional:*

--formBean

The finder's search parameter. Should be a DTO and it must have at least same fields (name and type) as those included in the finder **--name**, which can be target entity fields or related entity fields.

Possible values are: any of the DTO's in the project.

This option is mandatory if `--returnType` is specified and its a projection.

This option is not available if `--entity` parameter has not been specified before or if it does not exist any DTO in generated project.

Default if option not present: the entity specified in `--entity` option.

--returnType

The finder's results return type.

Possible values are: Projection classes annotated with `@RooEntityProjection` and related to the entity specified in `--entity` option (use `entity projection` command), or the same entity.

This option is not available if `--entity` parameter has not been specified before or if it does not exist any Projection class associated to the targeted entity.

Default if not present: the default return type of the repository related to the entity, which can be specified with `--defaultReturnType` parameter in `repository jpa` command.

focus

Changes Roo Shell focus to a different type in the project.

```
<strong>roo</strong> focus [--class]
```

- *Mandatory:*

--class

The type to focus on (mandatory). When working on a mono module project, simply specify the name of the class in which the new constant will be included. If you consider it necessary, you can also specify the package. Ex.: `--class ~.domain.MyEnumClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class model:~.domain.MyEnumClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

help

Shows a summary of all Spring Roo commands.

```
<strong>roo</strong> help [--command]
```

- *Optional:*

--command

Command name to provide help for. When command name has more than one word, it should be between quotation marks.

hint

Provides step-by-step hints and context-sensitive guidance.

```
<strong>roo</strong> hint [--topic]
```

- *Optional:*

--topic

The topic for which advice should be provided.

Possible values are: `controllers`, `eclipse`, `entities`, `fields`, `finders`, `general`, `mvc`, `persistence`, `relationships`, `repositories`, `services`, `start` and `topics`.

interface

Creates a new Java interface source file in any project path.

```
<strong>roo</strong> interface [--class] [--path --permitReservedWords --force]
```

- *Mandatory:*

--class

The name of the class to create. If you consider it necessary, you can also specify the package (base package can be specified with `~`). Ex.: `--class ~.domain.MyClass`. You can specify module as well, if necessary. Ex.: `--class model:~.domain.MyClass`. When working with a multi-module project, if module is not specified the class will be created in the module which has the focus.

- *Optional:*

--path

Source directory to create the interface in.

Default: `[FOCUSED-MODULE]/src/main/java`.

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: **true**; default if option not present: **false**.

--force

Force command execution.

Default if option present: **true**; default if option not present: **false**.

jms receiver

Creates an JMS receiver.

```
<strong>roo</strong> jms receiver [--destinationName --endpoint --jndiConnectionFactory]
[--profile --force]
```

- *Mandatory:*

--destinationName

The name of the JMS destination, composed by 'application' type module and destination name. If only have one 'application' type module or focused module is the 'application' module that you want to use, don't include it, only write destination name. If only have one 'application' type module or focused module is the 'application' module that you want to use, don't include it, only write destination name.

--endpoint

The service where include the method that receives JMS messages.

--jndiConnectionFactory

The jndi name for which the JMS receiver configuration has been defined.

- *Optional:*

--profile

The profile where the properties will be set.

--force

Force command execution.

Default if option present: **true**; default if option not present: **false**.

jms sender

Creates an JMS sender.

```
<strong>roo></strong> jms sender {--class --destinationName --jndiConnectionFactory} [--profile --force]
```

- *Mandatory:*

--class

The class where include a reference to the JMS which sends messages.

--destinationName

The name of the JMS destination, composed by 'application' type module and destination name. module and destination name. If only have one 'application' type module or focused module is the 'application' module that you want to use, don't include it, only write destination name.

--jndiConnectionFactory

The jndi name where the JMS sender configuration has been defined.

- *Optional:*

--profile

The profile where the properties will be set.

--force

Force command execution.

Default if option present: **true**; default if option not present: **false**.

jpa audit add

Adds support for auditing a JPA entity. This will add JPA and Spring listeners to this entity to record the entity changes.

```
<strong>roo></strong> jpa audit add {--entity} (--createdDateColumn --modifiedDateColumn --createdByColumn --modifiedByColumn)
```

- *Mandatory:*

--entity

The entity which should be audited. When working on a mono module project, simply specify the name of the entity. If you consider it necessary, you can also specify the package. Ex.: **--class ~.domain.MyEntity** (where ~ is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: **--class model:~.domain.MyEntity**. If the module is not specified, it is assumed that the entity is in the module which has the focus.

- *Conditional:*

--createdDateColumn

The DB column used for storing the date when each record is created.

This option is mandatory if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

--modifiedDateColumn

The DB column used for storing the date when each record is modified.

This option is mandatory if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

--createdByColumn

The DB column used for storing information about who creates each record.

This option is mandatory if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

--modifiedByColumn

The DB column used for storing information about who modifies each record.

This option is mandatory if `spring.roo.jpa.require.schema-object-name` configuration setting exists and it's `true`.

jpa audit setup

Installs audit support into your project, preparing it to audit entity changes.

```
<strong>roo</strong> jpa audit setup (--module)
```

- *Conditional:*

--module

The application module where to install the audit support.

This option is mandatory if the focus is not set in an application module, that is, a module containing an `@SpringBootApplication` class.

This option is available only if there are more than one application module and none of them is focused.

Default if option not present: the unique 'application' module, or focused 'application' module.

jpa setup

Installs or updates a JPA persistence provider in your project. User can execute this command for different profiles with different persistence configurations.

```
<strong>roo</strong> jpa setup {--provider} (--jndiDataSource | --database [--hostName  
--databaseName --userName --password]) (--module) [--force --profile]
```

- *Mandatory:*

--provider

The persistence ORM provider to support.

Possible values are: **ECLIPSELINK** and **HIBERNATE**.

This option is available only if **--jndiDataSource** has not been specified.

This option is mandatory if **--jndiDataSource** has not been specified.

- *Conditional:*

--database

The database type to support. Possible values are: **DB2_400**, **DB2_EXPRESS_C**, **DERBY_CLIENT**, **DERBY_EMBEDDED**, **FIREBIRD**, **H2_IN_MEMORY**, **HYPERSONIC_IN_MEMORY**, **HYPERSONIC_PERSISTENT**, **MSSQL**, **MYSQL**, **ORACLE**, **POSTGRES** and **SYBASE**.

This option is mandatory if **--jndiDataSource** has not been specified.

This option is available only if **--jndiDataSource** has not been specified.

--module

The application module where to install the persistence.

This option is mandatory if the focus is not set in an application module, that is, a module containing an **@SpringBootApplication** class.

This option is available only if there are more than one application module and none of them is focused.

Default if option not present: the unique 'application' module, or focused 'application' module.

--jndiDataSource

The JNDI datasource to use.

This option is not available if any of **--provider**, **--database**, **--databaseName**, **--hostName**, **--password** or **--userName** options are specified.

--hostName

The host name to use.

This option is available if `--database` has already been specified and its value is not `HYPERSONIC` or `H2_IN_MEMORY` and `--jndiDatasource` has not been specified.

--databaseName

The database name to use.

This option is available if `--database` has already been specified and its value is not `HYPERSONIC` or `H2_IN_MEMORY` and `--jndiDatasource` has not been specified.

--userName

The username to use.

This option is available if `--database` has already been specified and its value is not `HYPERSONIC` or `H2_IN_MEMORY` and `--jndiDatasource` has not been specified.

--password

The password to use.

This option is available if `--database` has already been specified and its value is not `HYPERSONIC` or `H2_IN_MEMORY` and `--jndiDatasource` has not been specified.

- *Optional:*

--force

Force command execution; default if option present: `true`; default if option not present: `false`

--profile

Parameter that indicates the name of the profile that will be applied.

module create

Creates a new Maven module in current **multimodule** project.

```
<strong>roo</strong> module create [--moduleName] [--packaging --artifactId]
```

- *Mandatory:*

--moduleName

The name of the module to create.

- *Optional:*

--packaging

The Maven packaging of this module.

Possible values are: **BUNDLE**, **EAR**, **ESA**, **JAR** and **WAR**.

Default if option not present: **JAR** (equals to 'jar').

--artifactId

The artifact ID of this module.

Default if option not present: **--moduleName** value.

module focus

Changes Roo Shell focus to a different project module, when in a multimodule project.

```
<strong>roo</strong> module focus {--moduleName}
```

- *Mandatory:*

--moduleName

The module name to focus on.

Possible values are: any of the project module names (~ for root module).

project setup

Creates a new Maven project.

```
<strong>roo</strong> project setup {--topLevelPackage} [--multimodule | --packaging] [--  
projectName --java]
```

- *Mandatory:*

--topLevelPackage

The uppermost package name (this becomes the **<groupId>** in Maven and also the ~ value when using Roo Shell).

- *Optional:*

--projectName

The name of the project (this becomes the **<artifactId>** in Maven).

Default if option not present: last segment of `--topLevelPackage` name used.

--multimodule

Option to use a multimodule architecture.

Possible values are: **BASIC** (root module with child 'application' module), and **STANDARD BASIC** (root module with child 'application' module), and (root module with children modules: 'application', 'model', 'repository', 'service-api', 'service-impl' and 'integration').

Default if option present: **STANDARD**

--java

Forces a particular major version of Java to be used.

Default if option not present: Java 6 inherited from Spring Boot.

--packaging

The Maven packaging of this project.

This option is not available if 'multimodule' is specified.

Default if option not present: 'jar'.

property add

Adds or updates a particular property from application config properties file.

```
<strong>roo</strong> property add {--key --value} (--module) [--force --profile]
```

- *Mandatory:*

--key

The property key that should be changed.

--value

The new value for this property key.

- *Conditional:*

--module

Module where property will be added.

This option is mandatory if the focus is not set in an application module, that is, a module containing an `@SpringBootApplication` class.

This option is available only if there are more than one application module and none of them is focused.

Default if option not present: the unique 'application' module, or focused 'application' module.

- *Optional:*

- force**

- Force command execution.

- Default if option present: **true**; default if option not present: **false**.

- profile**

- Parameter that indicates the name of the profile that will be applied.

property list

List all properties from application config properties file.

```
<strong>roo</strong> property list (--module) [--force --profile]
```

- *Conditional:*

- module**

- Module which properties will be listed.

- This option is mandatory if the focus is not set in an application module, that is, a module containing an **@SpringBootApplication** class.

- This option is available only if there are more than one application module and none of them is focused.

- Default if option not present: the unique 'application' module, or focused 'application' module.

- *Optional:*

- force**

- Force command execution.

- Default if option present: **true**; default if option not present: **false**

- profile**

- Parameter that indicates the name of the profile that will be applied.

property remove

Removes a particular property from application config properties file.

```
<strong>roo</strong> property remove [--key] (--module) [--force --profile]
```

- *Mandatory:*

--key

The property key that should be removed.

- *Conditional:*

--module

Module where property will be removed.

This option is mandatory if the focus is not set in an application module, that is, a module containing an `@SpringBootApplication` class.

This option is available only if there are more than one application module and none of them is focused.

Default if option not present: the unique 'application' module, or focused 'application' module.

- *Optional:*

--force

Force command execution.

Default if option present: `true`; default if option not present: `false`.

--profile

Parameter that indicates the name of the profile that will be applied.

push-in

Allows to push-in elements declared in the ITDs to its .java files. You could specify `--all` option to apply push-in on every component of generated project, or you could define any package, class or method to apply push-in, combining them.

```
<strong>roo</strong> push-in (--all | --package --class --method) [--force]
```

- *Conditional:*

--all

Option that indicates if push-in process should be applied to entire project.

This option is mandatory if none of **--package**, **--class** or **--method** are specified. Otherwise, using **--package**, **--class** or **--method** will cause the parameter **--all** won't be available.

--package

JavaPackage with the specified package where developer wants to make push-in. In multi-module project you should specify the module name before the package name. Ex.: **--package model:org.springframework.roo** but, if module name is not present, the Roo Shell focused module will be used.

This option is not available if **--all** parameter has been already specified.

--class

JavaType with the specified class where developer wants to make push-in. When working on a single module project, simply specify the name of the class. If you consider it necessary, you can also specify the package. Ex.: **--class ~.domain.MyClass** (where ~ is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: **--class model:~.domain.MyClass**. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is not available if **--all** parameter has been already specified.

--method

String with the specified name of the method which developer wants to push-in. You could use a Regular Expression to make push-in of more than one method on the same execution.

This option is not available if **--all** parameter has been already specified.

• *Optional:*

--force

Force command execution.

Default if option present: **true**; default if option not present: **false**.

quit

Waits until all metadata and files are refreshed and updated, then exits the shell. You can also use **exit** command.

```
<strong>roo</strong> quit
```

This command does not accept any options.

repository jpa

Generates new Spring Data repository for specified entity or for all entities in generated project.

```
roo repository jpa (--all [--package] | --entity --interface [--defaultReturnType])
```

- *Conditional:*

--all

Indicates if developer wants to generate repositories for every entity of current project.

This option is mandatory if **--entity** is not specified. Otherwise, using **--entity** will cause the parameter **--all** won't be available.

Default if option present: **true**; default if option not present: **false**.

--entity

The domain entity this repository should manage. When working on a single module project, simply specify the name of the entity. If you consider it necessary, you can also specify the package. Ex.: **--class ~.domain.MyEntity** (where ~ is the base package). When working with multiple modules, you should specify the name of the entity and the module where it is. Ex.: **--class model:~.domain.MyEntity**. If the module is not specified, it is assumed that the entity is in the module which has the focus.

Possible values are: any of the entities in the project.

This option is mandatory if **--all** is not specified. Otherwise, using **--all** will cause the parameter **--entity** won't be available.

--interface

The java Spring Data repository to generate. When working on a single module project, simply specify the name of the class. If you consider it necessary, you can also specify the package. Ex.: **--class ~.domain.MyClass** (where ~ is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: **--class model:~.domain.MyClass**. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is mandatory if **--entity** has been already specified and the project is multi-module.

This option is available only when **--entity** has been specified.

- *Optional:*

--defaultReturnType

The default return type which this repository will have for all finders, including those created by default. The default return type should be a Projection class associated to the entity specified in **--entity** parameter.

Possible values are: any of the projections associated to the entity in **--entity** option.

This option is not available if domain entity specified in **--entity** parameter has no associated Projections.

Default: the entity specified in the **entity** option.

--package

The package where repositories will be generated. In multi-module project you should specify the module name before the package name. Ex.: **--package model:org.springframework.roo** but, if module name is not present, the Roo Shell focused module will be used.

This option is not available if **--all** option has not been specified.

Default value if not present: **~.repository** package, or 'repository:~.' if multi-module project.

script

Parses the specified resource file and executes its Roo commands. You can as well execute ***roo** example scripts in the Roo classpath. Ex.: **script --file clinic.roo**.

```
<strong>roo</strong> script [--file] [--ignoreLines --lineNumbers]
```

- *Mandatory:*

--file

The file to locate and execute.

- *Optional:*

--ignoreLines

Comma-list of prefixes to ignore the lines that starts with any of the provided case-sensitive prefixes.

--lineNumbers

Display line numbers when executing the script.

Default if option present: **true**; default if option not present: **false**

security authorize

Includes **@PreAuthorize** annotation to an specific method for controlling access to its invocation.

```
<strong>roo</strong> security authorize [--class --method] (--roles | --usernames)
```

- *Mandatory:*

--class

The service class that contains the method to annotate with **@PreAuthorize**. When working on a single module project, simply specify the name of the class. If you consider it necessary, you can also specify the package. Ex.: **--class ~.service.MyClass** (where **~** is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: **--class service:~.MyClass**. If the module is not specified, it is assumed that the class is in the module which has the focus.

Possible values are: any of the service classes in the project.

--method

The service method name (including its params) that will be annotated with **@PreAuthorize**. Is possible to specify a regular expression.

Possible values are: any of the existing methods of the class specified in **--class** option, or regular expression.

- *Conditional:*

--roles

Comma separated list with all the roles to add inside 'hasAnyRole' instruction.

This option is mandatory if **--usernames** is not specified.

--usernames

Comma separated list with all the usernames to add inside Spring Security annotation.

This option is mandatory if **--roles** is not specified.

security filtering

Include **@PreFilter**/**@PostFilter** annotation to an specific method to filter results of a method invocation based on an expression.

```
<strong>roo</strong> security filtering [--class --method] (--roles | --usernames) [--when]
```

- *Mandatory:*

- class**

- The service class that contains the method to annotate. When working on a single module project, simply specify the name of the class. If you consider it necessary, you can also specify the package. Ex.: `--class ~.service.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class service:~.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

- Possible values are: any of the service classes in the project.

- method**

- The service method name (including its params), that will be annotated with `@PreFilter` / `@PostFilter`. Is possible to specify a regular expression.

- Possible values are: any of the existing methods of the class specified in `--class` option, or regular expression.

- *Conditional:*

- roles**

- Comma separated list with all the roles to add inside 'hasAnyRole' instruction.

- This option is mandatory if `--usernames` is not specified.

- usernames**

- Comma separated list with all the usernames to add inside Spring Security annotation.

- This option is mandatory if `--roles` is not specified.

- *Optional:*

- when**

- Indicates if filtering should be after or before to execute the operation. Depends of the specified value, `@PreFilter` annotation or `@PostFilter` annotation will be included.

- Possible values are: `PRE` and `POST`.

- Default: `PRE`.

security setup

Install Spring Security into your project.

```
<strong>roo</strong> security setup (--module) [--provider]
```

- *Conditional:*

--module

The application module where to install the security support.

This option is mandatory if the focus is not set in an application module, that is, a module containing an `@SpringBootApplication` class.

This option is available only if there are more than one application module and none of them is focused.

Default if option not present: the unique 'application' module, or focused 'application' module.

- *Optional:*

--provider

The Spring Security provider to install.

Possible values are: **DEFAULT** (default Spring Security configuration provided by Spring Boot will be used), and **SPRINGLETS_JPA** (advanced Spring Security configuration will be included using Springlets JPA Authentication).

service

Creates new service interface and its implementation related to an entity, or for all the entities in generated project, with some basic management methods by using Spring Data repository methods.

```
<strong>roo</strong> service (--all [--apiPackage --implPackage] | --entity --repository  
--interface [--class])
```

- *Conditional:*

--all

Indicates if developer wants to generate service interfaces and their implementations for every entity of current project.

This option is mandatory if **--entity** is not specified. Otherwise, using **--entity** will cause the

parameter `--all` won't be available.

Default if option present: `true`; default if option not present: `false`.

--entity

The domain entity this service should manage. When working on a single module project, simply specify the name of the entity. If you consider it necessary, you can also specify the package. Ex.: `--class ~.domain.MyEntity` (where `~` is the base package). When working with multiple modules, you should specify the name of the entity and the module where it is. Ex.: `--class model:~.domain.MyEntity`. If the module is not specified, it is assumed that the entity is in the module which has the focus.

Possible values are: any of the entities in the project.

This option is mandatory if `--all` is not specified. Otherwise, using `--all` will cause the parameter `--entity` won't be available.

--repository

The repository this service should expose. When working on a single module project, simply specify the name of the class. If you consider it necessary, you can also specify the package. Ex.: `--class ~.repository.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class repository:~.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

Possible values are: any of the repositories annotated with `@RooJpaRepository` and associated to the entity specified in `--entity`.

This option is mandatory if `--entity` has been already specified and the project is multi-module.

This option is available only when `--entity` has been specified.

Default if option not present: first repository annotated with `@RooJpaRepository` and associated to the entity specified in `--entity`.

--interface

The service interface to be generated. When working on a single module project, simply specify the name of the class. If you consider it necessary, you can also specify the package. Ex.: `--class ~.service.api.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class service-api:~.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is mandatory if `--entity` has been already specified and the project is multi-module.

This option is available only when `--entity` has been specified.

Default if option not present: concatenation of entity simple name with 'Service' in `~.service.api` package, or 'service-api:~.' if multi-module project.

- *Optional:*

--class

The service implementation to be generated. When working on a single module project, simply specify the name of the class. If you consider it necessary, you can also specify the package. Ex.: `--class ~.service.impl.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class service-impl:~.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

This option is available only when `--entity` has been specified.

Default if option not present: concatenation of entity simple name with 'ServiceImpl' in `~.service.impl` package, or 'service-impl:~.' if multi-module project.

--apiPackage

The java interface package. In multi-module project you should specify the module name before the package name. Ex.: `--apiPackage service-api:org.springframework.roo` but, if module name is not present, the Roo Shell focused module will be used.

This option is available only when `--all` parameter has been specified.

Default value if not present: `~.service.api` package, or 'service-api:~.' if multi-module project.

--implPackage

The java package of the implementation classes for the interfaces. In multi-module project you should specify the module name before the package name. Ex.: `--implPackage service-impl:org.springframework.roo` but, if module name is not present, the Roo Shell focused module will be used.

This option is available only when `--all` parameter has been specified.

Default value if not present: `~.service.impl` package, or 'service-impl:~.' if multi-module project.

settings add

Adds or updates a Roo project setting, which can modify the configuration of some commands acting in the current project. These settings are located in `[PROJECT-ROOT]/.roo/config/project.properties`.

```
<strong>roo</strong> settings add {--name --value} [--force]
```

- *Mandatory:*

--name

The setting name that should be added or changed.

--value

The value for this settings name.

- *Optional:*

--force

Force command execution.

Default if option present: **true**; default if option not present: **false**.

settings list

Lists all settings added into Roo project configuration. These settings are located in *[PROJECT-ROOT]/.roo/config/project.properties*.

```
<strong>roo</strong> settings list
```

This command does not accept any options.

settings remove

Removes a specific setting from Roo project configuration. Use 'settings list' to see the Roo settings added to the project.

```
<strong>roo</strong> settings remove {--name}
```

- *Mandatory:*

--name

The settings name that should be removed.

test integration

Creates a new integration test class for the specified class. The generated test class will contain a basic

structure and the necessary testing components.

```
<strong>roo</strong> test integration [--class] (--module) [--permitReservedWords]
```

- *Mandatory:*

--class

The name of the class to create an integration test. If you consider it necessary, you can also specify the package. Ex.: **--class** `~.package.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: **--class** `module:~.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

Possible values are: any of the valid classes in the project which support automatically integration test creation.

- *Conditional:*

--module

The application module where generate the integration test.

This option is mandatory if the focus is not set in an 'application' module and there are more than one 'application' modules, that is, a module containing an `@SpringBootApplication` class.

This option is available only if there are more than one application module and none of them is focused.

Default if option not present: the unique 'application' module, or focused 'application' module.

- *Optional:*

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: `true`; default if option not present: `false`.

test unit

Creates a unit test class with a basic structure and with the necessary testing components, for the specified class.

```
<strong>roo</strong> test unit [--class] [--permitReservedWords]
```

- *Mandatory:*

--class

The name of the project class which this unit test class is targeting. If you consider it necessary, you can also specify the package. Ex.: `--class ~/.model.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name" of the class and the module where it is. Ex.: `--class model:~.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

- *Optional:*

--permitReservedWords

Indicates whether reserved words are ignored by Roo.

Default if option present: `true`; default if option not present: `false`.

version

Displays Roo Shell banner and version.

```
<strong>roo</strong> version
```

This command does not accept any options.

web flow

Installs a Spring Web Flow into your project.

```
<strong>roo</strong> web flow [--flowName] (--module) [--class]
```

- *Mandatory:*

--flowName

The name for your web flow.

- *Conditional:*

--module

The application module where create the web flow.

This option is mandatory if the focus is not set in an 'application' module and there are more than one 'application' modules, that is, a module containing an `@SpringBootApplication` class.

This option is available only if there are more than one application module and none of them is focused.

Default if option not present: the unique 'application' module, or focused 'application' module.

- *Optional:*

--class

The class used to create the model object this flow is mainly bound to. Can be an entity or a DTO and must be serializable.

web mvc controller

Generates new `@RooController`'s in the directory `src/main/java` of the selected project module (if any). The generated controllers should manage specific entities in the project.

```
<strong>roo</strong> web mvc controller (--all | --entity ) [--responseType --package  
--pathPrefix]
```

- *Conditional:*

--all

Indicates if developer wants to generate controllers for every entity of current project.

This option is mandatory if `--entity` is not specified. Otherwise, using `--entity` will cause the parameter `--all` won't be available.

Default if option present: `true`; default if option not present: `false`.

--entity

The domain entity this controller should manage. When working on a single module project, simply specify the name of the entity. If you consider it necessary, you can also specify the package. Ex.: `--class ~.domain.MyEntity` (where `~` is the base package). When working with multiple modules, you should specify the name of the entity and the module where it is. Ex.: `--class model:~.domain.MyEntity`. If the module is not specified, it is assumed that the entity is in the module which has the focus.

Possible values are: any of the entities in the project.

This option is mandatory if `--all` is not specified. Otherwise, using `--all` will cause the parameter `--entity` won't be available.

- *Optional:*

--responseType

Indicates the responseType to be used by generated controller. Depending on the selected responseType, generated methods and views will vary.

Possible values are: **JSON** plus any response type installed with **web mvc view setup** command.

This option is available once **--all** or **--entity** parameters have been specified.

Default: **JSON**.

--package

Indicates which package should be used to include generated controllers. In multi-module project you should specify the module name before the package name. Ex.: **--package application:org.springframework.roo.web** but, if module name is not present, the Roo Shell focused module will be used.

This option is available only if **--all** or **--entity** option has been specified.

Default value if not present: **~.web** package, or 'application:~.web' if multi-module project.

--pathPrefix

Indicates **@RequestMapping** prefix to be used on this controller. It is not necessary to specify '/' as Spring Roo shell will include it automatically.

This option is available only if **--all** or **--entity** option has been specified.

web mvc detail

Generates new **@RooController** for relation fields which detail wants to be managed. It must be a **@OneToMany** field. Generated controllers will have **@RooDetail** with info about the parent entity and the parent views where the detail will be displayed.

```
<strong>roo</strong> web mvc detail (--all | --entity [--field]) [--package  
--responseType --views]
```

- *Conditional:*

--all

Indicates if developer wants to generate detail controllers for each **@OneToMany** relation of field in each entity in the project.

This option is mandatory if **--entity** is not specified. Otherwise, using **--entity** will cause the parameter **--all** won't be available.

Default if option present: **true**; default if option not present: **false**.

--entity

Indicates the entity which this detail controller manages. When working on a single module project, simply specify the name of the entity. If you consider it necessary, you can also specify

the package. Ex.: `--class ~.domain.MyEntity` (where `~` is the base package). When working with multiple modules, you should specify the name of the entity and the module where it is. Ex.: `--class model:~.domain.MyEntity`. If the module is not specified, it is assumed that the entity is in the module which has the focus.

Possible values are: any of the entities in the project.

This option is mandatory if `--all` is not specified. Otherwise, using `--all` will cause the parameter `--entity` won't be available.

- *Optional:*

--field

Indicates the entity's field on which the detail controller is generated. It must be a `@OneToMany` field.

Possible values are: fields representing a `@OneToMany` relation of the entity specified in `--entity` parameter.

This param is only available if `--entity` parameter has been specified before.

--package

Indicates the Java package where the detail controllers should be generated. In multi-module project you should specify the module name before the package name. Ex.: `--package application:org.springframework.roo.web` but, if module name is not present, the Roo Shell focused module will be used.

This option is available only if `--all` or `--entity` option has been specified.

Default if option not present: `~.web` package, or 'application:~.web' if multi-module project.

--responseType

Indicates the responseType to be used by generated detail controllers. Depending on the selected responseType, generated methods and views will vary.

Possible values are: `JSON` plus any response type installed with `web mvc view setup` command.

This option is available once `--all` or `--entity` parameters have been specified.

Default: `JSON`.

--views

Separated comma list where developer could specify the different parent views where this new detail will be displayed.

This parameter is not available if the provided `--responseType` doesn't use views to display the data.

Possible values are: 'list', 'show' or the different parent finder views (if exists).

Default if option not present: The parent 'list' view if it exists.

web mvc exception handler

Adds methods to handle an application exception in a specified controller or a class annotated with `@ControllerAdvice`.

```
<strong>roo</strong> web mvc exception handler [--exception] [--class --controller  
--errorView]
```

- *Mandatory:*

--exception

The exception to handle. If you consider it necessary, you can also specify the package. Ex.: `--class ~.model.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class model:~.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

- *Optional:*

--class

Class annotated with `@ControllerAdvice` where include the handler methods. If you consider it necessary, you can also specify the package. Ex.: `--class ~.model.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class model:~.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

--controller

Controller where include the handler methods. If you consider it necessary, you can also specify the package. Ex.: `--class ~.model.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: `--class model:~.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

--errorView

View to be returned when specified exception is thrown.

web mvc finder

Publishes existing finders to web layer, generating controllers and additional views for them. It adds

`@RooWebFinder` annotation to MVC controller type.

```
<strong>roo</strong> web mvc finder (--all | --entity [--queryMethod]) [--package  
--pathPrefix --responseType]
```

- *Conditional:*

--entity

The entity owning the finders that should be published. When working on a single module project, simply specify the name of the entity. If you consider it necessary, you can also specify the package. Ex.: `--class ~.domain.MyEntity` (where `~` is the base package). When working with multiple modules, you should specify the name of the entity and the module where it is. Ex.: `--class model:~.domain.MyEntity`. If the module is not specified, it is assumed that the entity is in the module which has the focus.

Possible values are: any of the entities in the project.

This option is mandatory if `--all` is not specified. Otherwise, using `--all` will cause the parameter `--entity` won't be available.

--all

Indicates if developer wants to publish in web layer all finders from all entities in project. This option is mandatory if `--entity` is not specified. Otherwise, using `--entity` will cause the parameter `--all` won't be available.

Default if option present: `true`; default if option not present: `false`.

- *Optional:*

--queryMethod

Indicates the name of the finder to add to web layer.

Possible values are: any of the finder names created for the entity, included in `@RooJpaRepository` of the `--entity` associated repository.

This option is available only when `--entity` has been specified.

--responseType

Indicates the responseType to be used by generated finder controllers. Depending on the selected responseType, generated methods and views will vary.

Possible values are: `JSON` plus any response type installed with `web mvc view setup` command.

This option is only available if `--all` or `--entity` parameters have been specified. Default: `JSON`.

--package

Indicates the Java package where the finder controllers should be generated. In multi-module project you should specify the module name before the package name. Ex.: **--package application:org.springframework.roo.web** but, if module name is not present, the Roo Shell focused module will be used.

This option is available only if **--all** or **--entity** option has been specified.

Default value if not present: **~.web** package, or 'application:~.web' if multi-module project.

--pathPrefix

Indicates the default path value for accessing finder resources in controller, used for this controller **@RequestMapping** excluding first '/'.

This option is available only if **--all** or **--entity** option has been specified.

web mvc language

Installs new language in generated project views. Also, could be used to specify the default language of the project.

```
<strong>roo</strong> web mvc language {--code} (--module) [--useAsDefault]
```

- *Mandatory:*

--code

The language code for the desired bundle.

Possible values are: supported languages. Currently **en** (English, default) and **es** (Spanish).

- *Conditional:*

--module

The application module where to install the language support.

This option is mandatory if the focus is not set in an application module, that is, a module containing an **@SpringBootApplication** class.

This option is available only if there are more than one application module and none of them is focused.

Default if option not present: the unique 'application' module, or focused 'application' module.

- *Optional:*

--useAsDefault

Indicates if selected language should be used as default on this application.

Default: `false`.

web mvc setup

Includes Spring MVC configuration on generated project. Needed for several MVC related commands.

```
<strong>roo</strong> web mvc setup (--module)
```

- *Conditional:*

--module

The application module where to install the Spring MVC support.

This option is mandatory if the focus is not set in an application module, that is, a module containing an `@SpringBootApplication` class. This option is available only if there are more than one application module and none of them is focused.

Default if option not present: the unique 'application' module, or focused 'application' module.

web mvc templates setup

Includes view generation templates on current project. Will allow developers to customize view generation by modifying the templates from `[PROJECT-ROOT]/.roo/templates/...`

```
<strong>roo</strong> web mvc templates setup [--type]
```

- *Mandatory:*

--type

View identifier of templates you want to install. Only installed views are available. Views can be installed with `web mvc view setup` command.

web mvc view setup

Includes all necessary resources of provided response type on generated project. This response type will be needed by `web mvc controller`, `web mvc detail` and `web mvc templates setup` commands.


```
<strong>roo</strong> web mvc view setup {--type} (--module)
```

- *Mandatory:*

--type

View identifier you want to install. This is known as 'responseType' in other **web mvc** commands.

- *Conditional:*

--module

The application module where to install views.

This option is mandatory if the focus is not set in an application module, that is, a module containing an **@SpringBootApplication** class.

This option is available only if there are more than one application module and none of them is focused.

Default if option not present: the unique 'application' module, or focused 'application' module.

ws client

Generates a new Web Service client by the provided WSDL file.

```
<strong>roo</strong> ws client {--wsdl --endpoint --class} [--binding --serviceUrl  
--profile]
```

- *Mandatory:*

--wsdl

WSDL file located in some specific module. By default, Spring Roo searches .wsdl files in the 'src/main/resources/' folder of the existing modules.

--endpoint

Select some endpoint defined in the .wsdl file provided before. This parameter will be autocompleted with the attribute 'name' of the 'port' element inside the 'service' element.

--class

Configuration class that will include the method to define the Web Service client. You could provide a new class.

- *Optional:*

--binding

The binding type to be used. You could choose between SOAP11 and SOAP12. If not specified, it will be calculated using the .wsdl file namespace.

--serviceUrl

The service URL to be used. If This option is not specified, default location provided by the .wsdl file will be used. This default location will be obtained from the 'location' attribute of the 'address' element located inside the 'port' element provided in the '--endpoint' parameter.

--profile

Parameter that indicates the name of the profile that will be applied.

ws endpoint

Generates a new Service Endpoint Interface (SEI) and its implementation.

```
<strong>roo</strong> ws endpoint [--service --sei] [--class --config --profile --force]
```

- *Mandatory:*

--service

Existing service annotated with **@RooService** that will be used to generate the new SEI. The new generated SEI will include all defined operations in the provided service interface.

Possible values are: any of the project service classes, annotated with **@RooService**.

--sei

New Service Endpoint Interface to generate. It's not possible to indicate an existing class.

- *Optional:*

--class

New class that will implement the new generated SEI. If not specified, a new implementation class will be generated in the same module using the SEI name and the 'Endpoint' suffix.

--config

Configuration class that will register the new endpoint. You could specify an existing **@Configuration** class or indicates a new one to be generated. If not specified, a new **@Configuration** class will be generated in the same module using the SEI name and the 'Configuration' suffix.

--profile

Parameter that indicates the name of the profile that will be applied.

--force

Force command execution.

Appendix B: Command index for add-on management

NOTE | [Command syntax](#)

addon info bundle

Provide information about a specific Spring Roo Add-on from installed repositories.

```
<strong>roo</strong> addon info bundle {--bundleSymbolicName}
```

- *Mandatory:*

--bundleSymbolicName

The bundle symbolic name of the add-on of interest.

addon install bundle

Install Spring Roo Add-on from installed repositories

```
<strong>roo</strong> addon install bundle {--bundleSymbolicName}
```

- *Mandatory:*

--bundleSymbolicName

The bundle symbolic name of the add-on of interest.

addon install url

Installs Spring Roo Add-on using an URL.

```
<strong>roo</strong> addon install url {--url}
```

- *Mandatory:*

--url

The url of the add-on of interest.

addon list

Lists all installed add-ons.

```
<strong>roo</strong> addon list
```

This command does not accept any options.

addon remove

Removes an installed Spring Roo Add-on.

```
<strong>roo</strong> addon remove {--bundleSymbolicName}
```

- *Mandatory:*

--bundleSymbolicName

The bundle symbolic name of the add-on of interest.

addon repository add

Adds a new OBR Repository to Roo Shell.

```
<strong>roo</strong> addon repository add {--url}
```

- *Mandatory:*

--url

URL file that defines repository. Ex: 'http://localhost/repo/index.xml'.

NOTE

See that in Windows systems, you must use **file:**\ protocol when you specify a local repository URL. However, in Unix systems the protocol for local repositories URL must be **file:**//.

addon repository introspect

Introspects all installed OBR Repositories and list all their add-ons.

```
<strong>roo</strong> addon repository introspect
```

This command does not accept any options.

addon repository list

Lists installed OBR Repositories.

```
<strong>roo</strong> addon repository list
```

This command does not accept any options.

addon repository remove

Removes an existing OBR Repository from Roo Shell.

```
<strong>roo</strong> addon repository remove {--url}
```

- *Mandatory:*

--url

URL file that defines repository. Ex: 'http://localhost/repo/index.xml'.

NOTE

See that in Windows systems, you must use **file:**\ protocol when you specify a local repository URL. However, in Unix systems the protocol for local repositories URL must be **file://**.

addon search

Searches all known Spring Roo Add-ons from installed repositories.

```
<strong>roo</strong> addon search {--requiresCommand}
```

- *Mandatory:*

--requiresCommand

Only display add-ons in search results that offer this command.

addon suite install name

Installs some 'Roo Addon Suite' from installed OBR Repository.

```
<strong>roo</strong> addon suite install name {--symbolicName}
```

- *Mandatory:*

--symbolicName

Name that identifies the 'Roo Addon Suite'.

addon suite install url

Installs some 'Roo Addon Suite' from URL.

```
<strong>roo</strong> addon suite install url {--url}
```

- *Mandatory:*

--url

URL of Roo Addon Suite .esa file.

addon suite list

Lists all installed 'Roo Addon Suite'. If you want to list all available 'Roo Addon Suites' on Repository, use **--repository** parameter.

```
<strong>roo</strong> addon suite list [--repository]
```

- *Optional:*

--repository

OBR Repository where the 'Roo Addon Suite' are located.

addon suite start

Starts some installed 'Roo Addon Suite'. By default, an installed 'Roo Addon Suite' is started automatically.

```
<strong>roo</strong> addon suite start {--symbolicName}
```

- *Mandatory:*

--symbolicName

Name that identifies the 'Roo Addon Suite'.

addon suite stop

Stops some started 'Roo Addon Suite'.

```
<strong>roo</strong> addon suite stop {--symbolicName}
```

- *Mandatory:*

--symbolicName

Name that identifies the 'Roo Addon Suite'.

addon suite uninstall

Uninstalls some installed 'Roo Addon Suite'.

```
<strong>roo</strong> addon suite uninstall {--symbolicName}
```

- *Mandatory:*

--symbolicName

Name that identifies the 'Roo Addon Suite'.

Appendix C: Command index for add-on development

These commands are specific for developing Spring Roo add-ons and will be only available if user enables the add-on "development mode" with `addon development mode` command.

NOTE | [Command syntax](#)

!g

Passes a command directly through to the Felix shell infrastructure

```
<strong>roo</strong> !g
```

- *Mandatory:*

--[default]

The command to pass to Felix (WARNING: no validation or security checks are performed); default: 'help'

addon create advanced

Create a new advanced add-on for Spring Roo (commands + operations + metadata + trigger annotation + dependencies).

```
<strong>roo</strong> addon create advanced [--topLevelPackage] [--description  
--projectName]
```

- *Mandatory:*

--topLevelPackage

The top level package of the new addon. In Maven, this will be the `<groupId>`.

- *Optional:*

--description

Description of your addon (surround text with double quotes).

--projectName

Provide a custom project name. In Maven, this will be the `<artifactId>`.

Default if option not present: the top level package specified in `--topLevelPackage`.

addon create i18n

Create a new internationalization add-on for Spring Roo, with a new language. Created add-on can be installed later into a project for localizing the project to that new language.

```
<strong>roo</strong> addon create i18n [--topLevelPackage --locale --messageBundle] [--  
language --flagGraphic --description --projectName]
```

- *Mandatory:*

- topLevelPackage**

- The top level package of all Spring Roo Addon Suite. In Maven, this will be the `<groupId>`.

- locale**

- The locale abbreviation (ie: en, or more specific like en_AU, or de_DE) for the new language.

- messageBundle**

- Fully qualified path to the messages_xx.properties file which contains the messages in the new language.

- *Optional:*

- language**

- The full name of the language (used as a label for the UI).

- flagGraphic**

- Fully qualified path to new flag xx.png file.

- description**

- Description of your addon (surround text with double quotes).

- projectName**

- Provide a custom project name. In Maven, this will be the `<artifactId>`.

- Default if option not present: the top level package specified in `--topLevelPackage`.

addon create simple

Create a new simple add-on for Spring Roo (commands + operations).

```
<strong>roo</strong> addon create simple {--topLevelPackage} [--description  
--projectName]
```

- *Mandatory:*

--topLevelPackage

The top level package of the new addon. In Maven, this will be the `<groupId>`.

- *Optional:*

--description

Description of your addon (surround text with double quotes).

--projectName

Provide a custom project name. In Maven, this will be the `<artifactId>`.

Default if option not present: the top level package specified in `--topLevelPackage`.

addon create suite

Create a new Spring Roo Addon Suite for Spring Roo (two sample addons + repository + suite generator).

```
<strong>roo</strong> addon create suite {--topLevelPackage} [--description  
--projectName]
```

- *Mandatory:*

--topLevelPackage

The top level package of all Spring Roo Addon Suite. In Maven, this will be the `<groupId>`.

- *Optional:*

--description

Description of your Roo Addon Suite (surround text with double quotes).

--projectName

Provide a custom project name for root module. In Maven, this will be the `<artifactId>`.

Default if option not present: the top level package specified in `--topLevelPackage`.

addon create wrapper

Create a new add-on for Spring Roo which wraps a maven artifact to create a OSGi compliant bundle.

```
<strong>roo</strong> addon create wrapper {--topLevelPackage --groupId --artifactId  
--version --vendorName --licenseUrl} [--docUrl --description --projectName --osgiImports]
```

- *Mandatory:*

- topLevelPackage**

- The top level package of the new wrapper bundle.

- groupId**

- Dependency group id.

- artifactId**

- Dependency artifact id.

- version**

- Dependency version.

- vendorName**

- Dependency vendor name.

- licenseUrl**

- Dependency license URL.

- *Optional:*

- docUrl**

- Dependency documentation URL.

- description**

- Description of the bundle (use keywords with #-tags for better search integration).

- projectName**

- Provide a custom project name. In Maven, this will be the `<artifactId>`.

- Default if option not present: the top level package specified in `--topLevelPackage`.

- osgiImports**

- Contents of Import-Package in OSGi manifest.

addon development mode

Switches the system into development mode, which enables add-on development commands and shows greater diagnostic information.

```
<strong>roo</strong> addon development mode [--enabled]
```

- *Optional:*

- **--enabled**

Activates addon development mode.

Default: **true**

metadata cache

Shows detailed metadata for the indicated type.

```
<strong>roo</strong> metadata cache {--maximumCapacity}
```

- *Mandatory:*

- **--maximumCapacity**

The maximum number of metadata items to cache.

metadata for id

Shows detailed information about the metadata item.

```
<strong>roo</strong> metadata for id {--metadataId}
```

- *Mandatory:*

- **--metadataId**

The metadata ID (should start with MID:).

metadata for module

Shows the ProjectMetadata for the indicated project module.

```
<strong>roo</strong> metadata for module [--module]
```

- *Optional:*

--module

The module for which to retrieve the metadata.

Default if option not present: the Roo Shell focused module.

metadata for type

Shows detailed metadata for the indicated type

```
<strong>roo</strong> metadata for type [--type]
```

- *Mandatory:*

--type

The Java type for which to display metadata. When working on a single module project, simply specify the name of the class. If you consider it necessary, you can also specify the package. Ex.: **--type** `~.domain.MyClass` (where `~` is the base package). When working with multiple modules, you should specify the name of the class and the module where it is. Ex.: **--type** `model:~.domain.MyClass`. If the module is not specified, it is assumed that the class is in the module which has the focus.

metadata status

Shows metadata statistics of the current project.

```
<strong>roo</strong> metadata status
```

This command does not accept any options.

metadata trace

Traces metadata event delivery notifications.

```
<strong>roo</strong> metadata trace [--level]
```

- *Mandatory:*

--level

The verbosity of notifications (0=none, 1=some, 2=all).

process manager debug

Indicates if process manager debugging is desired. It is only available if 'addon development mode' is true.

```
<strong>roo</strong> process manager debug [--enabled]
```

- *Optional:*

--enabled

Activates debug mode, which shows status of process manager such as 'Active' or 'Scanning'.

Default: `true`.

project scan now

Performs a manual file system scan, calling thread monitors and checking that all files are updated.

```
<strong>roo</strong> project scan now
```

This command does not accept any options.

project scan speed

Changes the time interval between file system scans.

```
<strong>roo</strong> project scan speed [--ms]
```

- *Mandatory:*

--ms

The number of milliseconds between each scan.

project scan status

Displays file system scanning information such as the time lasted for last scan and scanning frequency.

```
<strong>roo</strong> project scan status
```

This command does not accept any options.

reference guide

Writes the reference guide XML fragments (in DocBook format) into the current working directory. It is only available if 'development mode' is **true**.

```
<strong>roo</strong> reference guide
```

This command does not accept any options.

system properties

Shows the shell's properties such as if 'addon development mode' is enabled, JVM version, file encoding...

```
<strong>roo</strong> system properties
```

This command does not accept any options.

Appendix D: Using Spring Roo without IDE

Installing Spring Roo

Once you have met the initial [requirements](#), you are ready to install Roo by following these steps:

1. Download the current release from Spring Roo project page [downloads section](#).

NOTE | You can also build a distribution ZIP yourself from our [source control repository](#).

2. Unzip the distribution, which will unpack to a single installation directory; this will be known as **\$ROO_HOME** in the paths below.

- If using Windows, add **\$ROO_HOME\bin** to your **PATH** environment variable
- If using Linux or MacOS, create a symbolic link using a command such as:

```
$ sudo ln -s $ROO_HOME/bin/roo.sh /usr/bin/roo
```

3. Next verify Roo has been installed correctly. This can be done, using the following commands:

```
$ mkdir roo-test
$ cd roo-test
$ roo
```

```

      _
  _  ( )  _
 /  _  \  _  \  _  \  _  \  _  \  _  \  _  \  _  \  _  \
 \  _  \  _  \  _  \  _  \  _  \  _  \  _  \  _  \  _  \
 |  _  \  _  \  _  \  _  \  _  \  _  \  _  \  _  \  _  \
   |  _  \  _  \  _  \  _  \  _  \  _  \  _  \  _  \  _  \
                                     W.X.Y.ZZ
```

```
Welcome to Spring Roo. For assistance press TAB or type "hint" then hit ENTER.
roo>
```

If Roo logo appears, you have installed Roo successfully. For those curious, the "[rev RRR]" refers to the Git commit ID used to compile that particular build of Roo.

Backup and deployment

A very useful command is the [backup](#) command. Using this command you will create a backup of the current workspace with all sources, log files and the script log file (excluding the target directory):

```
<strong>roo></strong> backup
```

Finally, you may wish to deploy your application to a production Web container. For this you can easily create two war files, by using the Maven command `mvn package` in the project base directory. You can perform this as well without exiting from Roo shell by using `!os` command, providing you have Maven in the `$PATH` variable:

```
<strong>roo></strong> !os mvn package
```

This command generates a `"*.war"` file which can then be easily copied into your production Web container and a `"*exec.war"` file that uses an embedded web server.

You can execute `"*exec.war"` as follows:

```
$ java -jar name-exec.war
```

NOTE | The provider dependencies are added only in `"*exec.war"` file.

Appendix E: Roo Resources

As an open source project, Spring Roo offers a large number of resources to assist the community learn, interact with one another and become more involved in the project. Below you'll find a short summary of the official project resources.

Spring Roo Project Home Page

The definitive source of information about Spring Roo is the [Spring Roo Home](http://spring.io) at <http://spring.io>.

That site provides a brief summary of Roo's main features and links to most of the other project resources. The project home page serves as a hub of information and is the best place to find up-to-date announcements about the project as well as links to articles, blogs and new documentation.

Please use this URI if you are referring other people to the Spring Roo project, as it is the main landing point for the project.

Downloads and Maven Repositories

You can always access the latest Spring Roo release ZIP by visiting Downloads section at [Spring Roo Home Page](#).

We publish all Roo modules to Maven Central, the default repository from which Maven will download the Spring Roo artifacts automatically.

StackOverflow

Because Roo is an official top-level Spring project, of course you'll find there is a dedicated "Spring Roo" tag at Stack Overflow for all your questions, comments and experiences.

If you have any question about Spring Roo project and its functionalities, you can check and ask at [Spring Roo tagged questions at Stack Overflow](#). We monitor stackoverflow.com for questions tagged with spring-roo.

<http://forum.springsource.org> is now a read-only archive. All commenting, posting, registration services have been turned off.

The Roo project does not have a "mailing list" or "newsgroup" as you might be familiar with from other open source projects, although [commercial support](#) options are available.

Extensive search facilities are provided on the community forums, and the Roo developers routinely answer user questions. One excellent way of contributing to the Roo project is to simply keep an eye on the forum messages and help other people. Even recommendations along the lines of, "I don't know how to do what you're trying to do, but we usually tackle the problem this way instead...." are very

helpful to other community members.

When you ask a question on the forum, it's highly recommended you include a small Roo [sample script](#) that can be used to reproduce your problem. If that's infeasible, using Roo's "[backup](#)" command is another alternative and you can attach the resulting ZIP file to your post. Other tips include always specifying the version of Roo that you're running (as can be obtained from the "[version](#)" command), and if you're having trouble with IDE integration, the exact version of the IDE you are using (and, if an Eclipse-based IDE, the version of [AspectJ Development Tools](#) in use). Another good source of advice on how to ask questions on the forum can be found in Eric Raymond's often-cited essay, "[How to Ask Smart Questions](#)".

If you believe you have found a bug or are experiencing an issue, it is recommended you first log a message on the forum. This allows other experienced users to comment on whether it appears there is a problem with Roo or perhaps just needs to be used a different way. Someone will usually offer a solution or recommend you log a bug report (usually by saying "please log this in Jira"). When you do log a bug report, please ensure you link to the fully-qualified URI to the forum post. That way the developer who attempts to solve your bug will have background information. Please also post the issue tracking link back in thread you started on the forum, as it will help other people cross-reference the two systems.

Twitter

Roo Hash Code (please include in your tweets, and also follow for low-volume announcements): [#SpringRoo](#)

If you use Twitter, you're encouraged to follow [@SpringRoo](#). Also please use [@SpringRoo](#) in your tweets so everyone can easily see them.

The Roo team also uses and monitors tweets that include [#SpringRoo](#), so if you're tweeting about Roo, please remember to include [#SpringRoo](#) somewhere in the tweet. If you like Roo or have found it helpful on a project, please tweet about it and help spread the word!

Follow the core Roo development team for interesting Roo news and progress (higher volume than just following [@SpringRoo](#), but only a few Tweets per week): [@disid_corp](#), [@juanCaFX](#), [@enrique_ruiz_](#).

Many people who use Roo also use Twitter, including the core Roo development team. If you're a Twitter user, you're welcome to follow the Roo development team (using the Twitter IDs above) to receive up-to-the-minute Tweets on Roo activities, usage and events.

We do request that you use the [StackOverFlow](#) if you have a question or issue with Roo, as 140 characters doesn't allow us to provide in-depth technical support or provide a growing archive of historical answers that people can search against.

Issue Tracking

Web: <https://jira.spring.io/browse/ROO/>

Spring projects use Atlassian Jira for tracking bugs, improvements, feature requests and tasks. Roo uses a public Jira instance you're welcome to use in order to log issues, watch existing issues, vote for existing issues and review the changes made between particular versions.

As discussed in the [StackOverFlow](#) section, we ask that you refrain from logging bug reports until you've first discussed them on stackoverflow. This allows others to comment on whether a bug actually exists. When logging an issue in Jira, there is a field explicitly provided so you can link the forum discussion to the Jira issue.

Please note that every commit into the Roo [source repository](#) will be prefixed with a particular Jira issue number. All Jira issue numbers for the Roo project commence with "ROO-", providing you an easy way to determine the rationale of any change.

Because open source projects receive numerous enhancement requests, we generally prioritise enhancements that have patches included, are quick to complete or those which have received a large number of votes. You can vote for a particular issue by logging into Jira (it's fast, easy and free to create an account) and click the "vote" link against any issue. Similarly you can monitor the progress on any issue you're interested in by clicking "watch".

Enhancement requests are easier to complete (and therefore more probable to be actioned) if they represent fine-grained units of work that include as much detail as possible. Enhancement requests should describe a specific use case or user story that is trying to be achieved. It is usually helpful to provide a Roo [sample script](#) that can be used to explain the issue. You should also consider whether a particular enhancement is likely to appeal to most Roo users, and if not, whether perhaps writing it as an add-on would be a good alternative.

Source Repository

Read repository: <https://github.com/spring-projects/spring-roo.git>

The Git source control system is currently used by Roo for mainline development.

Historical releases of Roo can be accessed by browsing the tags branches within our Git repository. The mainline development of Roo occurs on the "master" branch.

"gh-pages" branch is used to build and publish Spring Roo's project page site based on Jekyll and GitHub Pages.

Commercial Products and Services

Web: <http://www.disid.com>

DISID Corporation employs the Roo development team and offers a wide range of products and professional services around Roo and the technologies which Roo enables. Available professional services include software factory, geographic information systems, web application development, mobile application development, training, consulting and mentoring. Please visit the above URI to

learn more about DISID products and services.

Web: <http://spring.io/>

Pivotal Software offers a wide range of products and professional services around Roo and the technologies which Roo enables. Available professional services include training, consulting, design reviews and mentoring, with products including service level agreement (SLA) backed support subscriptions, certified builds, indemnification and integration with various commercial products. Please visit the above URI to learn more about SpringSource products and services and how these can add value to your build-run-manage application lifecycle.

Other

Please let us know if you believe it would be helpful to list any other resources in this documentation.