

# **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. Requirements .....	6
4. Install Spring Roo .....	7
Using Spring Roo .....	9
5. The Roo shell .....	10
6. Impatient beginners .....	12
7. Create your Spring Boot application .....	13
8. Configure the project settings .....	14
9. Setup the persistence engine .....	15
10. The domain model .....	16
JPA entities .....	16
DTOs .....	20
11. The data access layer .....	21
Spring Data repositories .....	21
Default queries .....	21
12. The service layer .....	22
Service API and Impl .....	22
13. The view layer .....	23
Thymeleaf view engine .....	23
Spring MVC Controllers .....	23
Spring Webflow .....	25
14. The integration layer .....	26
REST API .....	26
WS API .....	26
Email .....	26
15. The security layer .....	28
Auditing JPA entities .....	28
16. The infrastructure layer .....	29
17. Customize the code .....	30
18. Javadoc in AsciiDoc .....	32
19. Running the application .....	33
Appendix .....	33
Appendix A: Command index for application development .....	34

!os .....	34
backup .....	34
cache setup .....	34
class .....	35
constructor .....	36
dto .....	36
embeddable .....	37
entity jpa .....	38
entity projection .....	40
enum constant .....	41
enum type .....	42
equals .....	43
exit .....	43
field boolean .....	44
field date .....	46
field embedded .....	48
field enum .....	49
field file .....	51
field list .....	52
field number .....	56
field other .....	58
field reference .....	60
field set .....	62
field string .....	66
finder add .....	68
focus .....	69
help .....	69
hint .....	70
interface .....	70
jpa audit add .....	71
jpa audit setup .....	72
jpa setup .....	72
module create .....	73
module focus .....	73
project setup .....	74
property add .....	74
property list .....	75
property remove .....	75
push-in .....	76
repository jpa .....	76
script .....	77
security authorize .....	77
security filtering .....	78
security setup .....	78

service .....	79
settings add .....	80
settings list .....	80
settings remove .....	80
test integration .....	80
test unit .....	81
web mvc controller .....	81
web mvc detail .....	82
web mvc finder .....	83
web mvc language .....	83
web mvc setup .....	84
web mvc templates setup .....	84
web mvc view setup .....	84
version .....	85
Appendix B: Command index for add-on management .....	86
addon info bundle .....	86
addon install bundle .....	86
addon install url .....	86
addon list .....	87
addon remove .....	87
addon repository add .....	87
addon repository introspect .....	87
addon repository list .....	88
addon repository remove .....	88
addon search .....	88
addon suite install name .....	88
addon suite install url .....	89
addon suite list .....	89
addon suite start .....	89
addon suite stop .....	89
addon suite uninstall .....	90
Appendix C: Command index for add-on development .....	91
!g .....	91
addon create advanced .....	91
addon create i18n .....	91
addon create simple .....	92
addon create suite .....	92
addon create wrapper .....	93
addon development mode .....	94
metadata cache .....	94
metadata for id .....	94
metadata for module .....	94
metadata for type .....	95
metadata status .....	95

metadata trace .....	95
process manager debug .....	95
project scan now .....	96
project scan speed .....	96
project scan status .....	96
reference guide .....	96
system properties .....	97
Appendix D: Using Spring Roo without IDE .....	98
Installing Spring Roo .....	98
Backup and deployment .....	98
Appendix E: Roo Resources .....	100
Spring Roo Project Home Page .....	100
Downloads and Maven Repositories .....	100
StackOverFlow .....	100
Twitter .....	101
Issue Tracking .....	101
Source Repository .....	102
Commercial Products and Services .....	102
Other .....	103

2.0.0.M3

© 2016 *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 **kdb:[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:



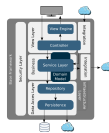
- [Apache CXF](#)
- [Springlets](#)
- ...

## 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 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 definition.
- 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. 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 3.0](#) 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 4. 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:

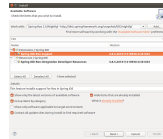
Version	STS update site
2.0.0.M3	Spring Roo 2.0 (Nightly)

Version	STS update site
2.0.0.RC1	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** ].

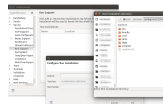
7. Restart the STS IDE

## Configure Spring Roo 2.0.0

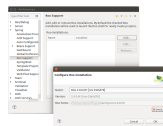
1. Open **Window** | **Preferences** | **Spring** | **Roo Support**.

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.



# 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.

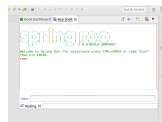
But first, let us to introduce the Roo shell.

# Chapter 5. 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.
- *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 6. 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
```

TODO source location at GitHub

## **Shop application with REST services**

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

TODO source location at GitHub

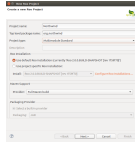
## **The classic Pet Clinic application (one Maven module project)**

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

TODO source location at GitHub

# Chapter 7. 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.

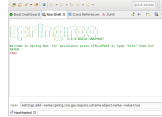
As you can see, the project extends the Spring IO platform, and it also adds the *spring boot starter* and the *spring boot starter test* dependencies.

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 8. 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 9. 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 10. 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 ~.CustomerOrder  
    entity jpa --class ~.OrderDetail  
    entity jpa --class ~.Party  
    entity jpa --class ~.Product  
    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
```

4. Add the attributes to the entites:

## 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 --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

```

## 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 11. The data access layer

## Spring Data repositories

```
<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 12. The service layer

## Service API and Impl

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

# Chapter 13. 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
web mvc detail --entity model:~.Category --responseType THYMELEAF --field
products.purchaseOrders
web mvc detail --entity model:~.Product --field purchaseOrders --responseType
THYMELEAF
web mvc detail --entity model:~.Country --responseType THYMELEAF --field regions
web mvc detail --entity model:~.Region --responseType THYMELEAF --field cities
```

## 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 14. 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 :: roo> email sender setup --service service-  
impl:~.CustomerServiceImpl --username USERNAME --password PASSWORD --host HOST --port  
PORT --protocol PROTOCOL --starttls true
```

## Receive email

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



# Chapter 15. 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 16. 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 repository:~.CustomerOrderRepository
    test unit --class service-api:~.CustomerOrderService
    test unit --class service-impl:~.CustomerServiceImpl

    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
```

# Chapter 17. 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:

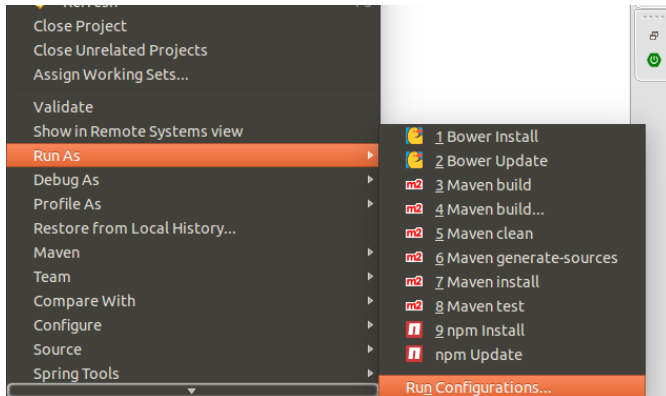
```
<strong>roo</strong> push-in --all --force
```

# Chapter 18. Javadoc in AsciiDoc

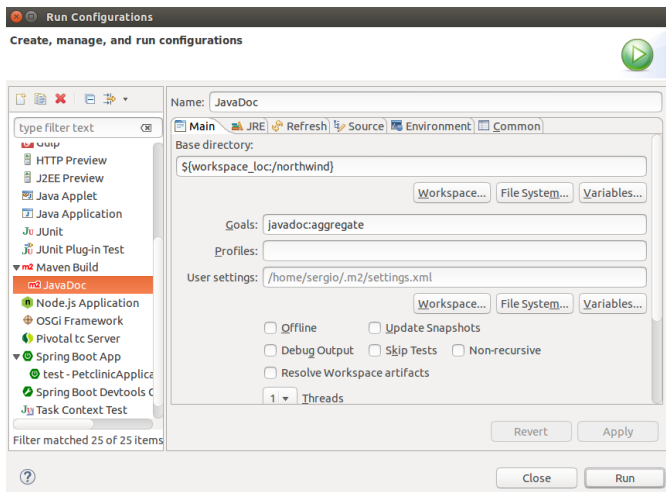
Spring Roo generated projects automatically include the "maven-javadoc-plugin" to generate project documentation following AsciiDoc syntax. This configuration it's done by using "[Asciidoclet](#)".

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.

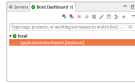


6. Apply configuration and close the window, or execute it directly with *Run*.
7. The generated JavaDoc will be in "[*ROOT-PROJECT*]/target/site/apidocs/".

# Chapter 19. 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

Commands are listed in alphabetic order, and are shown in monospaced font with any mandatory options you must 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.

## !os

Allows execution of operating system (OS) commands.

```
roo> !os
```

### --command

The command to execute; default: "

## backup

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

```
roo> 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.

```
roo> cache setup
```

- *Optional:*

### --provider

Parameter that indicates the provider to use for managing intermediate memory.

### --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.

```
roo> class --class
```

- *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:*

## **--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`.

## **--path**

Source directory to create the class in.

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

## **--extends**

The superclass fully qualified name.

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

## **--implements**

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`.

## **--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**.

## **constructor**

Creates a class constructor

```
roo> constructor
```

- *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.

```
roo> dto --class
```

- *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:*

### **--immutable**

Whether the DTO should be immutable.

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**.

### **--serializable**

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

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**.

## **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).

```
roo> embeddable --class
```

- *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.

```
roo> entity jpa --class
```

- *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 its value is `true`.

### **--table**

The JPA table name to use for this entity.

### **--identifierColumn**

The JPA identifier field column to use for this entity.

### **--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.

**--sequenceName**

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

**--identifierStrategy**

The generation value strategy to be used.

Default if option present: **AUTO**.

- *Optional:*

**--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**.

### **--equals**

Whether the generated class should implement equals and hashCode methods.

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`.

```
roo> entity projection [--all | --class --entity --fields]
```

- Mandatory on 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.

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

- *Conditional:*

### **--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'.

- *Optional:*

### **--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.

```
roo> enum constant --name
```

- *Mandatory:*

**--name**

The name of the constant. It will be 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

```
roo> enum type --class
```

- *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.

```
roo> equals
```

- *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.

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**

Exits the shell. You can also use **quit** command.

```
roo> exit
```

This command does not accept any options.



# field boolean

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

```
roo> field boolean --fieldName
```

- *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:*

## **--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`.

### **--nullRequired**

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

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

### **--assertFalse**

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

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.

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.

```
roo> field date --fieldName --type
```

- *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.

## **--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`

## **--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.

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.

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.

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.

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`.

### **--dateFormat**

Indicates the style of the date format (ignored if `dateTimeFormatPattern` is specified), adding `style` attribute to `org.springframework.format.annotation.DateTimeFormat` annotation into the field.

Possible values are: MEDIUM (style="MS"), NONE (style="-S") and SHORT (style="SS").

Default: MEDIUM.

### **--timeFormat**

Indicates the style of the time format (ignored if `dateTimeFormatPattern` is specified), adding `style` attribute to `org.springframework.format.annotation.DateTimeFormat` annotation into the field.

Possible values are: MEDIUM (style="MS"), NONE (style="-S") and SHORT (style="SS").

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.

### **--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.

```
roo> field embedded --fieldName --type
```

- *Mandatory:*

### **--fieldName**

The name of the field to add.

### **--type**

The Java type of an embeddable class, 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.

### **--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.

```
roo> field enum --fieldName --type
```

- *Mandatory:*

### **--fieldName**

The name of the field to add.

### **--type**

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

- *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.

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.

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.

```
roo> field file --fieldName --class --contentType --column
```

- *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.

```
roo> field list --fieldName --type
```

- *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.

## **--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`.

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

## **--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 for JPA entities and embeddable classes 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 for JPA entities and embeddable classes when `--joinTable` option is set.

### **--inverseJoinColumn**

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 for JPA entities and embeddable classes 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 for JPA entities and embeddable classes when `--joinTable` option is set.

### **--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.

### **--cardinality**

The relationship cardinality at a JPA level. This option is only available for JPA entities and embeddable classes.

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`. 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.

- *Optional:*

### **--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**.

### **--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.

### **--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**.

### **--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.

```
roo> field number --fieldName --type
```

- *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.

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.

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.

```
roo> field other --fieldName --type --class --column
```

- *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.

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.

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.

```
roo> field reference --fieldName --type
```

- *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 and embeddable classes.

### **--referencedColumnName**

The JPA `@JoinColumn referencedColumnName` attribute.

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

### **--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:*

### **--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`.

### **--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`.

### **--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`.

### **--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.

```
roo> field set --fieldName --type
```

- *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.

## **--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`.

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

## **--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 for JPA entities and embeddable classes 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 for JPA entities and embeddable classes 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 for JPA entities and embeddable classes 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 for JPA entities and embeddable classes when `--joinTable` option is set.

### **--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.

### **--cardinality**

The relationship cardinality at a JPA level. This option is only available for JPA entities and embeddable classes.

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`. 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.

- *Optional:*

### **--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`.

### **--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.

### **--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`.

### **--nullRequired**

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

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 string

Adds a private string field to an existing Java source file

```
roo> field string --fieldName
```

- *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:*

### **--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.

### **--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`.

### **--nullRequired**

Whether this value must be null. Adds `javax.validation.constraints.Null` 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.

### **--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.

```
roo> finder add --entity --name
```

- *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.

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.

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 entity specified in `--entity`.

## focus

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

```
roo> 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.

```
roo> help
```

- *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.

```
roo> hint
```

- *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.

```
roo> interface --class
```

- *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`.

## 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.

```
roo> jpa audit add --entity
```

- *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.

```
roo> jpa audit setup
```

- *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

Install or updates a JPA persistence provider in your project

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

### **--provider**

The persistence provider to support (mandatory)

### **--database**

The database to support (mandatory)

### **--module**

The application module where to install the persistence. This option is available if there is more than one application module (mandatory if the focus is not set in application module); default if option not present: ''

### **--jndiDataSource**

The JNDI datasource to use. This option is not available if any of `databaseName`, `hostName`, `password` or `userName` options are specified, or you are using an 'HYPERSONIC' or 'H2\_IN\_MEMORY' database.

### **--hostName**

The host name to use. Parameter database must be defined. Not available if jndiDatasource is specified or you are using an 'HYPERSONIC' or 'H2\_IN\_MEMORY' database.

#### **--databaseName**

The database name to use. Parameter database must be defined. Not available if jndiDatasource is specified or you are using an 'HYPERSONIC' or 'H2\_IN\_MEMORY' database.

#### **--userName**

The username to use. Parameter database must be defined. Not available if jndiDatasource is specified or you are using an 'HYPERSONIC' or 'H2\_IN\_MEMORY' database

#### **--password**

The password to use. Parameter database must be defined. Not available if jndiDatasource is specified or you are using an 'HYPERSONIC' or 'H2\_IN\_MEMORY' database

#### **--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.

```
roo> module create --moduleName
```

#### **--moduleName**

The name of the module (mandatory)

#### **--packaging**

The Maven packaging of this module; default if option not present: 'jar'

#### **--artifactId**

The artifact ID of this module; defaults: moduleName if not specified

## **module focus**

Changes focus to a different project module

```
roo> module focus --moduleName
```

**--moduleName**

The module to focus on (mandatory)

## project setup

Creates a new Maven project

```
roo> project setup --topLevelPackage
```

**--topLevelPackage**

The uppermost package name (this becomes the <groupId> in Maven and also the '~' value when using Roo's shell) (mandatory)

**--projectName**

The name of the project; default: last segment of package name used

**--multimodule**

Option to use a multimodule architecture; default if option present: 'STANDARD'

**--java**

Forces a particular major version of Java to be used; default: 8

**--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.

```
roo> property add --key --value --module
```

**--key**

The property key that should be changed (mandatory)

**--value**

The new value for this property key (mandatory)

**--module**

Module where property will be added. Not available if there is only one application module (mandatory if the focus is not set in application module); default if option not present: '.'

**--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.

```
roo> property list --module
```

**--module**

Module which properties will be listed. Not available if there is only one application module (mandatory if the focus is not set in application module); default if option not present: ''

**--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.

```
roo> property remove --key --module
```

**--key**

The property key that should be removed (mandatory)

**--module**

Module where property will be removed. Not available if there is only one application module (mandatory if the focus is not set in application module); default if option not present: ''

**--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

Push-in all methods, fields, annotations, imports, extends, etc.. declared on ITDs to its .java files. You could specify `--all` parameter to apply push-in on every component of generated project, or you could define package, class or method where wants to apply push-in.

```
roo> push-in
```

### **--all**

Parameter that indicates if push-in process should be applied to entire project. If specified, 'package', 'class' or 'method' parameters will be unavailable. This option is available if 'package', 'class' and 'method' parameters have not specified. It doesn't allow any value.

### **--package**

JavaPackage with the specified package where developers wants to make push-in. This option is available if 'all' parameter is not specified.

### **--class**

JavaType with the specified class where developer wants to make push-in. This option is available if 'all' parameter is not specified.

### **--method**

String with the specified name of the method that 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 available if 'all' parameter is not specified.

### **--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

## repository jpa

Generates new Spring Data repository for specified entity.

```
roo> repository jpa --interface
```

### **--all**

Indicates if developer wants to generate repositories for every entity of current project. Not available if 'entity' parameter has been specified before; default if option present: **true**; default if

option not present: **false**

### **--interface**

The java Spring Data repository to generate. Not available if 'entity' parameter has not been specified before (mandatory if 'entity' parameter has been specified and you are working under multimodule project)

### **--entity**

The domain entity this repository should expose. Not available if 'all' parameter has been specified before

### **--defaultReturnType**

The findAll finder return type. Should be a Projection class associated to the entity specified in 'entity' parameter. This option is not available if domain entity specified in 'entity' parameter has no associated Projections

### **--package**

The package where repositories will be generated. Not available if 'all' parameter has not been specified before

## **script**

Parses the specified resource file and executes its commands

```
roo> script --file
```

### **--file**

The file to locate and execute (mandatory)

### **--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**

Include @PreAuthorize annotation to an specific method.

```
roo> security authorize --class --method
```

**--class**

The service class that contains the method to annotate with `@PreAuthorize` (mandatory)

**--method**

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

**--roles**

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

**--usernames**

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

## security filtering

Include `@PreFilter/@PostFilter` annotation to an specific method.

```
roo> security filtering --class --method
```

**--class**

The service class that contains the method to annotate with `@PreFilter/@PostFilter` (mandatory)

**--method**

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

**--roles**

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

**--usernames**

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

**--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.; default: 'PRE'

## security setup

Install Spring Security into your project

```
roo> security setup --module
```

### **--type**

The Spring Security provider to install.; default: 'DEFAULT'

### **--module**

The application module where to install the persistence. Not available if there is only one application module (mandatory if the focus is not set in application module); default if option not present: ''

## **service**

Creates new service interface and its implementation.

```
roo> service --repository --interface
```

### **--all**

Indicates if developer wants to generate service interfaces and their implementations for every entity of current project. Not available if 'entity' parameter has been specified before; default if option present: **true**; default if option not present: **false**

### **--entity**

The domain entity this service should expose. Not available if 'all' parameter has been specified before

### **--repository**

The repository this service should expose. Not available if you don't specify 'entity' parameter (mandatory if multimodule project)

### **--interface**

The service interface to be generated. Not available if you don't specify 'entity' parameter (mandatory if multimodule project)

### **--class**

The service implementation to be generated. Not available if you don't specify 'entity' parameter

### **--apiPackage**

The java interface package. Not available if 'all' parameter has not been specified before

### **--implPackage**

The java package of the implementation classes for the interfaces. Not available if 'all' parameter has not been specified before

## settings add

Adds or updates a particular setting

```
roo> settings add --name --value
```

### **--name**

The setting name that should be changed (mandatory)

### **--value**

The new value for this (mandatory)

### **--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

## settings list

Lists all settings added into configuration

```
roo> settings list
```

This command does not accept any options.

## settings remove

Removes an specific setting from configuration

```
roo> settings remove --name
```

### **--name**

The settings name that should be removed (mandatory)

## test integration

Creates a new integration test for the specified entity

```
roo> test integration
```

### **--entity**

The name of the entity to create an integration test; default if option not present: '\*'

### **--permitReservedWords**

Indicates whether reserved words are ignored by Roo; default if option present: **true**; default if option not present: **false**

### **--transactional**

Indicates whether the created test cases should be run withing a Spring transaction; default: **true**

## **test unit**

Creates a unit test class for the specified class

```
roo> test unit --class
```

### **--class**

The name of the project class which this unit test class is targeting (mandatory)

### **--permitReservedWords**

Indicates whether reserved words are ignored by Roo; default if option present: **true**; default if option not present: **false**

## **web mvc controller**

Generates new @RooController's inside current project. The controllers should manage specific entities in the project.

```
roo> web mvc controller
```

### **--all**

Indicates if developer wants to generate controllers for every entity of current project. This param will be visible if 'entity' parameter has not been specified; default if option present: **true**; default if option not present: **false**

### **--entity**

Indicates the entity that new controller will manage. This param will be visible if 'all' parameter has not been specified

### **--responseType**

Indicates the responseType to be used by generated controller. Depending of the selected responseType, generated methods and views will vary. This param will be visible if 'all' or 'entity' parameters have been specified; default: 'JSON'

### **--package**

Indicates which package should be used to include generated controllers. This param will be visible if 'all' or 'entity' parameters have been specified

### **--pathPrefix**

Indicates @RequestMapping prefix to be used on this controller. Is not necessary to specify '/'. Spring Roo shell will include it automatically. This param will be visible if 'all' or 'entity' parameters have been specified; default: ""

## **web mvc detail**

Generates new @RooController's 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.

```
roo> web mvc detail
```

### **--all**

Indicates if developer wants to generate first detail controllers for every entity that has a controller of current project. This param will be visible if 'entity' parameter has not been specified; default if option present: **true**; default if option not present: **false**

### **--entity**

Indicates the entity on which the detail controller is generated. This param will be visible if 'all' parameter has not been specified

### **--field**

Indicates the entity's field on which the detail controller is generated. It must be a @OneToMany field. This param will be visible if 'entity' parameter has been specified before; default: ""

### **--package**

Indicates which package has the controllers on which the detail controllers are generated. This param will be visible if 'all' or 'entity' parameters have been specified

### **--responseType**

Indicates the responseType to be used by generated controller. Depending of the selected responseType, generated methods and views will vary. This param will be visible if 'all' or 'entity' parameters have been specified; default: 'JSON'

## web mvc finder

Adds @RooWebFinder annotation to MVC controller type

```
roo> web mvc finder --package
```

### --entity

The entity owning the finders that should be published. Not available if 'all' parameter has been specified before

### --all

Indicates if developer wants to publish in web layer all finders from all entities in project. Not available if 'entity' parameter has been specified before; default if option present: **true**; default if option not present: **false**

### --queryMethod

Indicates the name of the finder to add to web layer. Not available if 'entity' parameter has not been specified before

### --responseType

Indicates the responseType to be used by generated controller. Depending of the selected responseType, generated methods and views will vary. Not available if 'all' or 'entity' parameters have not been specified before

### --package

Indicates the package where generated controller will be located. If multimodule project, package must be in an application module (those with @SpringBootApplication class). Not available if 'all' or 'entity' parameters have not been specified before (mandatory if project has more than one 'application' modules) ; default if option not present: '~.web'

### --pathPrefix

Indicates the default path value for accesing finder resources in controller, excluding first '/'. Not available if 'all' or 'entity' parameters have not been specified before; default: ''

## web mvc language

Install new language in generated project. Also, could be used to specify the default language of the project.

```
roo> web mvc language --code --module
```



**--code**

The language code for the desired bundle (mandatory)

**--useAsDefault**

Indicates if selected language should be used as default on this application. By default false.; default: `false`

**--module**

The application module where to install message bundles. This option is not available if there is only one application module (mandatory if the focus is not set in application module); default if option not present: ''

## web mvc setup

Includes Spring MVC on generated project

```
roo> web mvc setup --module
```

**--module**

The application module where to install the persistence. This option is available if there is more than one application module (mandatory if the focus is not set in application module); default if option not present: ''

**--appServer**

The server where deploy the application; default if option not present: 'EMBEDDED'

## web mvc templates setup

Includes view generation templates on current project. Will allow developers to customize view generation.

```
roo> web mvc templates setup --type
```

**--type**

View identifier of templates you want to install. Only installed views are available (mandatory)

## web mvc view setup

Includes all necessary resources of provided responseType on generated project

```
roo> web mvc view setup --type --module
```

### **--type**

View identifier you want to install. Install your necessary views before to be used on controller generation command (mandatory)

### **--module**

The application module where to install views. Not available if there is only one application module (mandatory if the focus is not set in application module); default if option not present: ''

## **version**

Displays shell version

```
roo> version
```

### **--[default]**

Special version flags

# Appendix B: Command index for add-on management

## addon info bundle

Provide information about a specific Spring Roo Add-on from installed repositories.

```
roo> addon info bundle --bundleSymbolicName
```

- *Mandatory:*

**--bundleSymbolicName**

The bundle symbolic name of the add-on of interest.

## addon install bundle

Installs Spring Roo Add-on.

```
roo> addon install bundle --bundleSymbolicName
```

- *Mandatory:*

**--bundleSymbolicName**

The bundle symbolic name of the add-on of interest from installed repositories.

## addon install url

Installs Spring Roo Add-on using an URL.

```
roo> addon install url --url
```

- *Mandatory:*

**--url**

The url of the add-on of interest.

## addon list

Lists all installed add-ons.

```
roo> addon list
```

This command does not accept any options.

## addon remove

Removes an installed Spring Roo Add-on.

```
roo> 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.

```
roo> 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 nix systems the protocol for local repositories URL must be file://.

## addon repository introspect

Introspects all installed OBR Repositories and list all their add-ons.

```
roo> addon repository introspect
```

This command does not accept any options.

## addon repository list

Lists installed OBR Repositories.

```
roo> addon repository list
```

This command does not accept any options.

## addon repository remove

Removes an existing OBR Repository from Roo Shell.

```
roo> addon repository remove --url
```

- *Mandatory:*

**--url**

URL file that defines repository. Ex: 'http://localhost/repo/index.xml'.

## addon search

Searches all known Spring Roo Add-ons from installed repositories.

```
roo> 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.

```
roo> 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.

```
roo> 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.

```
roo> addon suite list
```

- *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.

```
roo> addon suite start --symbolicName
```

- *Mandatory:*

**--symbolicName**

Name that identifies the 'Roo Addon Suite'.

## addon suite stop

Stops some started 'Roo Addon Suite'.

```
roo> addon suite stop --symbolicName
```

- *Mandatory:*

**--symbolicName**

Name that identifies the 'Roo Addon Suite'.

## addon suite uninstall

Uninstall some installed 'Roo Addon Suite'.

```
roo> 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.

## !g

Passes a command directly through to the Felix shell infrastructure

```
roo> !g
```

### --[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)

```
roo> addon create advanced --topLevelPackage
```

### --topLevelPackage

The top level package of the new addon (mandatory)

### --description

Description of your addon (surround text with double quotes)

### --projectName

Provide a custom project name (if not provided the top level package name will be used instead)

## addon create i18n

Create a new Internationalization add-on for Spring Roo

```
roo> addon create i18n --topLevelPackage --locale --messageBundle
```



**--topLevelPackage**

The top level package of the new addon (mandatory)

**--locale**

The locale abbreviation (ie: en, or more specific like en\_AU, or de\_DE) (mandatory)

**--messageBundle**

Fully qualified path to the messages\_xx.properties file (mandatory)

**--language**

The full name of the language (used as a label for the UI)

**--flagGraphic**

Fully qualified path to flag xx.png file

**--description**

Description of your addon (surround text with double quotes)

**--projectName**

Provide a custom project name (if not provided the top level package name will be used instead)

## addon create simple

Create a new simple add-on for Spring Roo (commands + operations)

```
roo> addon create simple --topLevelPackage
```

**--topLevelPackage**

The top level package of the new addon (mandatory)

**--description**

Description of your addon (surround text with double quotes)

**--projectName**

Provide a custom project name (if not provided the top level package name will be used instead)

## addon create suite

Create a new Spring Roo Addon Suite for Spring Roo (two sample addons repository + suite generator)

```
roo> addon create suite --topLevelPackage
```

**--topLevelPackage**

The top level package of all Spring Roo Addon Suite (mandatory)

**--description**

Description of your Roo Addon Suite (surround text with double quotes)

**--projectName**

Provide a custom project name (if not provided the top level package name will be used instead)

## addon create wrapper

Create a new add-on for Spring Roo which wraps a maven artifact to create a OSGi compliant bundle

```
roo> addon create wrapper --topLevelPackage --groupId --artifactId --version --vendorName  
--licenseUrl
```

**--topLevelPackage**

The top level package of the new wrapper bundle (mandatory)

**--groupId**

Dependency group id (mandatory)

**--artifactId**

Dependency artifact id (mandatory)

**--version**

Dependency version (mandatory)

**--vendorName**

Dependency vendor name (mandatory)

**--licenseUrl**

Dependency license URL (mandatory)

**--docUrl**

Dependency documentation URL

**--description**

Description of the bundle (use keywords with #-tags for better search integration)

**--projectName**

Provide a custom project name (if not provided the top level package name will be used instead)

**--osgiImports**

Contents of Import-Package in OSGi manifest

## addon development mode

Switches the system into addon development mode, getting a greater diagnostic information and gaining access to addon development commands.

```
roo> addon development mode
```

**--enabled**

Activates addon development mode; default: **true**

## metadata cache

Shows detailed metadata for the indicated type

```
roo> metadata cache --maximumCapacity
```

**--maximumCapacity**

The maximum number of metadata items to cache (mandatory)

## metadata for id

Shows detailed information about the metadata item

```
roo> metadata for id --metadataId
```

**--metadataId**

The metadata ID (should start with MID:) (mandatory)

## metadata for module

Shows the ProjectMetadata for the indicated project module

```
roo> metadata for module
```

### **--module**

The module for which to retrieve the metadata ; default: the focused module

## **metadata for type**

Shows detailed metadata for the indicated type

```
roo> metadata for type --type
```

### **--type**

The Java type for which to display metadata (mandatory)

## **metadata status**

Shows metadata statistics

```
roo> metadata status
```

This command does not accept any options.

## **metadata trace**

Traces metadata event delivery notifications

```
roo> metadata trace --level
```

### **--level**

The verbosity of notifications (0=none, 1=some, 2=all) (mandatory)

## **process manager debug**

Indicates if process manager debugging is desired. It is only available if 'development mode' is true.

```
roo> process manager debug
```

### **--enabled**

Activates debug mode; default: `true`

## **project scan now**

Scan the project directory looking for artifacts that have changed since the last scan and if any, update the internal metadata cache.

```
roo> project scan now
```

This command does not accept any options.

## **project scan speed**

Changes the file system scanning frequency.

```
roo> project scan speed --ms
```

### **--ms**

The number of milliseconds between each scan (mandatory)

## **project scan status**

Display scanning process information.

```
roo> 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.

```
roo> reference guide
```

This command does not accept any options.

# system properties

Shows the shell's properties

```
roo> system properties
```

This command does not accept any options.



```
roo> backup
```

Finally, you may wish to deploy your application to a production Web container. For this you can easily create two war files, by taking advantage of the [perform package command](#):

```
roo> perform 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 a 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 a questions 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.