

Command

Plugin Documentation



Table of Contents

1. Introduction	1
2. Version History	2
3. Getting Started	3
3.1. Example application	3
4. The @ErrorHandler Annotation.....	4
Example Usage:	4
Code injected by the transformation into the affected actions:	5
Default error handler injected into all controllers:.....	6

Chapter 1. Introduction

The Command Plugin gives Grails a convention for command objects by adding a new artefact type. It also adds an AST transformation to eliminate some of the boilerplate for dealing with errors from command objects.

Chapter 2. Version History

- 1.0.1
 - Fixing issues:
 - [Improvement use respond instead of as JSON](#)
 - [create-command block built in rails create-command](#)
 - James Kleeh contributed some documentation changes.
- 1.0
 - Initial Release including the ErrorHandler annotation.

Chapter 3. Getting Started

1. Add the following dependency: .build.gradle

```
compile "org.grails.plugins:command:1.0.1"
```

1. Run grails create-command:

```
grails create-command-object <package>.<commandName>
```

or for the default package

```
grails create-command-object <commandName>
```

1. If this is the first command you've created, refresh the Gradle build. This will add the command folder to the source set. You can manually add the folder to the source set by right clicking on the folder, selecting "Mark Directory as", and selecting "Sources Root". In a future release I'll look for a better way to automate, and eliminate this step.

3.1. Example application

[GitHub testCommand](#)

Chapter 4. The @ErrorHandler Annotation

The @ErrorHandler annotation injects a call to the default error handling method, which is injected by the plugin. The annotation can either be applied to the controller class itself or to individual actions. If the annotation is applied at the class level it will be injected to each action, however applying it at the action level will override the class level behavior. The annotation can also be passed an optional name of an alternate method to call. The error handling functionality will not be applied to private methods or methods annotated with @SkipErrorHandler.

If you use parameters outside of a command object, and those parameters have binding errors, those will be included in the list sent to the error handler, but for each parameter you will have to include an entry in your i18n message bundle. For example:

```
params.<Your parameter name here>.conversion.error = Your error message for <Your  
parameter name> had an error binding.
```

Example Usage:

```

package test.command

import com.virtualdogbert.ast.ErrorsHandler
import grails.converters.JSON

@ErrorsHandler
class TestController {

    def index(TestCommand test) {
        //some controller code
    }

    @ErrorsHandler(handler = 'someOtherErrorHandler') //over rides the default.
    def list(TestCommand test) {
        //some controller code
    }

    @SkipErrorsHandler //Skips the error handler injection from the class
    annotation.
    def list(TestCommand test) {
        //some controller code
    }

    //Your error handler
    private boolean someOtherErrorHandler(List commandObjects) {
        List errors = commandObjects.inject([]) { result, commandObject ->

            if (commandObject.hasErrors()) {
                result + (commandObject.errors as JSON)
            } else {
                result
            }

        } as List

        if (errors) {
            //Do something
            return true
        }
        //possibly do something else
        return false
    }
}

```

Code injected by the transformation into the affected actions:

```

    if(errorsHandler( [<every commandObject in the actions parameter list>] )){ return
    null }

```

Default error handler injected into all controllers:

```
boolean errorHandler(List commandObjects) {
    List errors = commandObjects.inject([]) { result, commandObject ->

        if (commandObject.hasErrors()) {
            result + (commandObject.errors)
        } else {
            result
        }

    } as List

    if (errors) {
        response.status = Holders.config.getProperty('command.response.code',
Integer, 409)
        respond errors, [formats:['json', 'xml']]
        return true
    }

    return false
}
```

You can override the response code by setting `command.response.code` int your configuration.