Spring MVC - 1.7

On this page:

Target audience:

Users of the extension providing **Spring MVC** support for Web Services.

Summary: This document provides basic information about the extension providing **Spring MVC** support for Web Services.

Extension ID

com.castsoftware.springmvc

What's new?

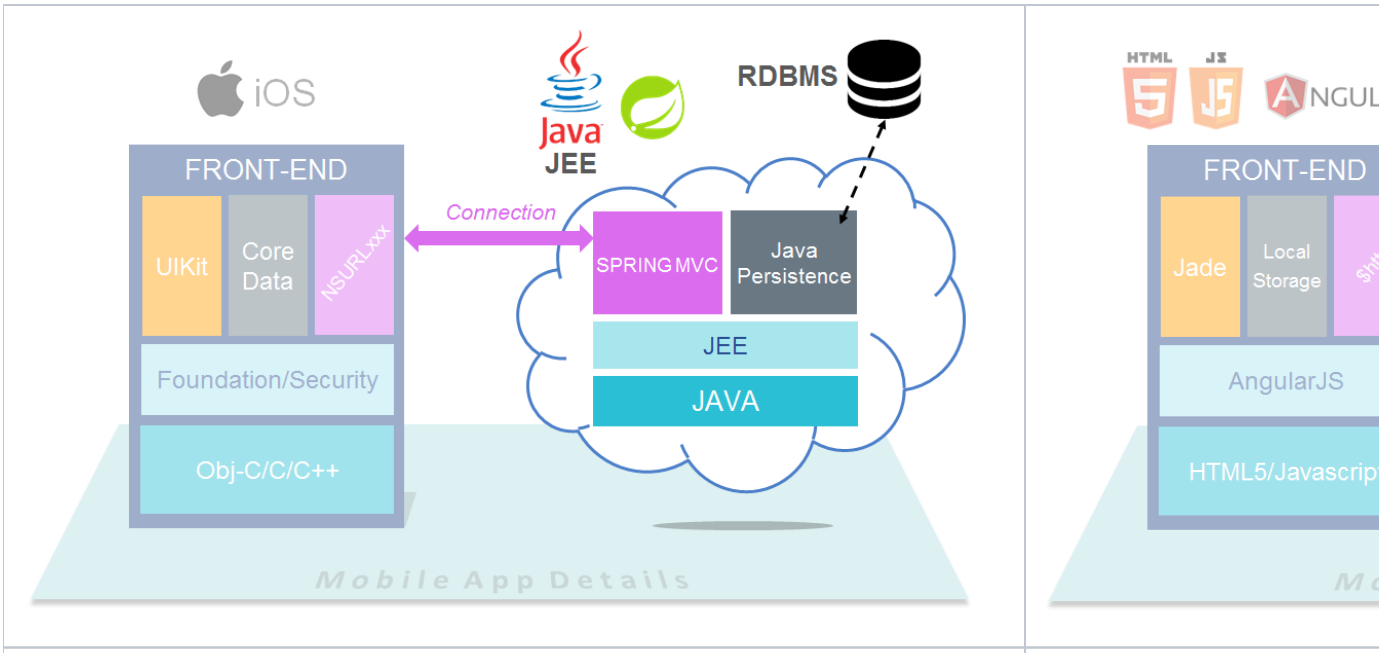
Please see Spring MVC 1.7 - Release Notes for more information.

Description

This extension provides support for **Spring MVC**.

In what situation should you install this extension?

The main purpose of this extension is to create **HTTP API entry points for JEE back-end applications** that are used to create **REST API Services**. If you need to have links from a **Client Front-end** (see examples below) to the **Spring MVC/JEE Back-end** you should install this extension.



iOS Front-End Example

Introduction

The SpringMVC framework is responsible of receiving all incoming HTTP requests and processing them by passing the requests to the relevant java component (controllers). The Spring analyzer will create Spring MVC Operation objects to represent these web services and links to Java objects to represent the posterior processing.

In SpringMVC, the rules mapping requests (URLs) to particular actions (methods inside controllers) can be configured with the help of handler mappings. The most common are:

Handler mapping	Supported
RequestMappingHandlerMapping	✅
BeanNameUrlHandlerMapping	✅
SimpleUrlHandlerMapping	✅
ControllerClassNameHandlerMapping	✅

RequestMappingHandlerMapping uses annotations to configuration web services configuration. The rest are usually configured via .xml files.

The supported controllers are: AbstractController, BaseCommandController, AbstractFormController, SimpleFormController, CancellableFormController, AbstractWizardFormController, AbstractCommandController,
AbstractUrlViewController, UrlFilenameViewController, ParameterizableViewController, ServletForwardingController, ServletWrappingController, MultiActionController (we highlight in red the deprecated controllers as of Spring 3.0, in favor of annotated controllers).

We further support mapping via resolution of method names and query parameters (methodNameResolver, and ParameterMethodNameResolver).

Features

This extension handles Spring MVC Web Services used in JEE applications, for example:

```
@RequestMapping("/users")
public class UserController {

    @RequestMapping(method = RequestMethod.GET, produces = "application/json; charset=utf-8")
    @ResponseBody
    public PagedResources<UserResource> collectionList(...){
        ...
    }
```

For each class annotated with

- **org.springframework.web.bind.annotation.RequestMapping (@RequestMapping)**
- **org.springframework.web.bind.annotation.GetMapping (@GetMapping)**

- **org.springframework.web.bind.annotation.PostMapping (@PostMapping)**
- **org.springframework.web.bind.annotation.PutMapping (@PutMapping)**

- **org.springframework.web.bind.annotation.DeleteMapping (@DeleteMapping)**
- **org.springframework.web.bind.annotation.PatchMapping (@PatchMapping)**,

the following will be done:

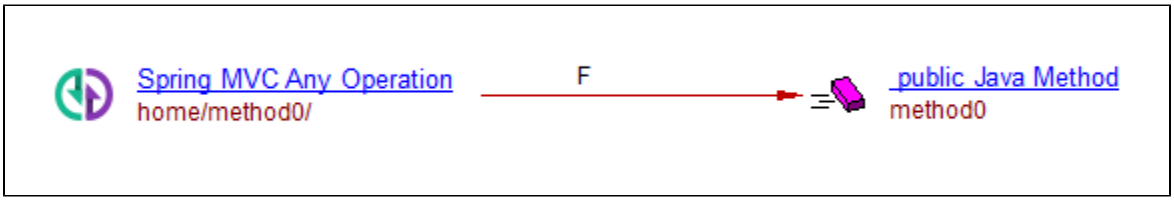
- a Web Service object will be created
- a Web Service Port object child of the web service will be created
- for each Method annotated with "Annotation"Mapping, the following will be created:
    - a Web Service operation child of the Web Service Port with correct get/put/delete/post type
    - a fire link from the Web Service operation to the method

Basic case @RequestMapping

The following Java code

```
@RequestMapping("/home")
public class HomeController {

    @RequestMapping(value="/method0")
    @ResponseBody
    public String method0(){
        return "method0";
    }

}
```

will generate:

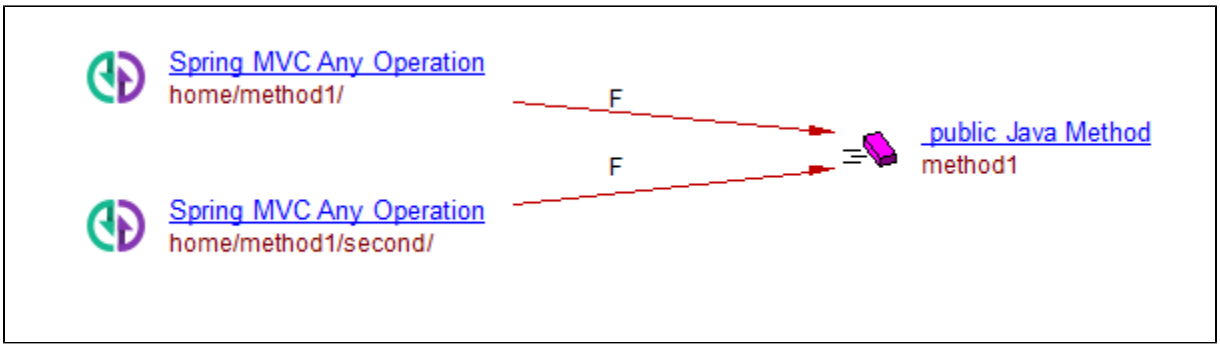


Several urls

The following Java code:

```
@RequestMapping("/home")
public class HomeController {

    @RequestMapping(value={"/method1","/method1/second"})
    @ResponseBody
    public String method1(){
        return "method1";
    }

}
```

will generate one operation per url mapping:

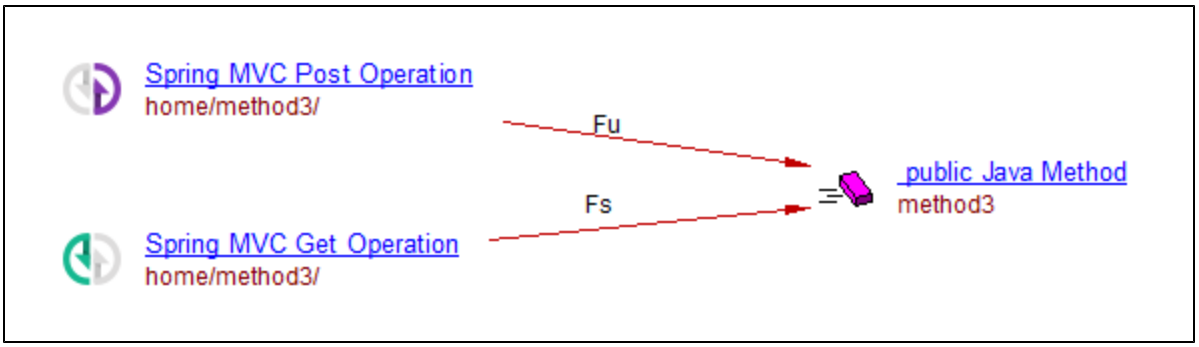


Several methods

The following Java code:

```
@RequestMapping("/home")
public class HomeController {

    @RequestMapping(value="/method3", method={RequestMethod.POST,RequestMethod.GET})
    @ResponseBody
    public String method3(){
        return "method3";
    }
}
```

will generate one operation per receiving method:

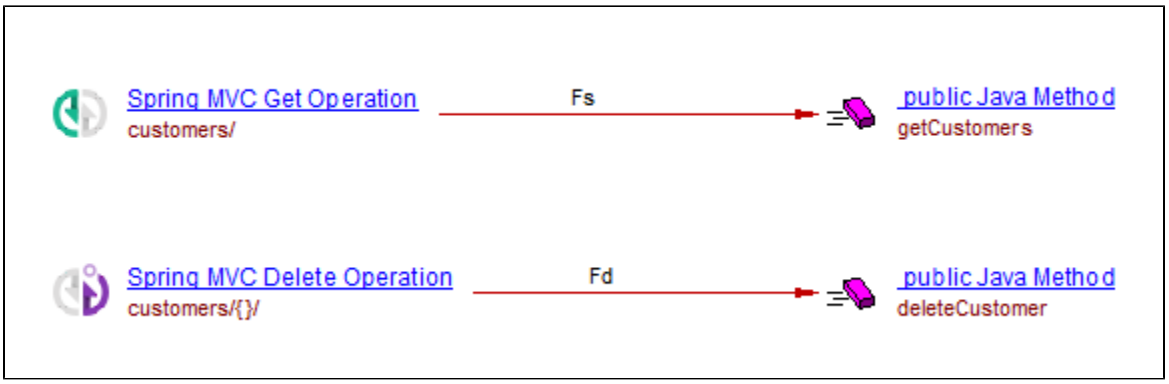


Spring 4 annotations

The syntax @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping are supported. Example with the following Java code:

```
@RestController
public class CustomerRestController {

    @GetMapping("/customers")
    public List getCustomers() {
        // ...
    }

    @DeleteMapping("/customers/{id}")
    public ResponseEntity deleteCustomer(@PathVariable Long id) {
        // ...
    }
}
```

will generate:



Property evaluation

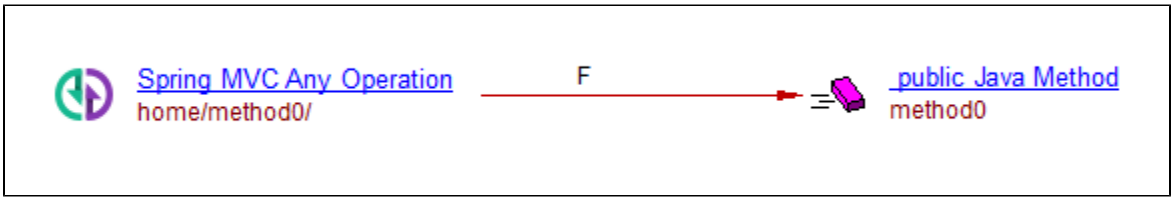
Properties used inside annotations are evaluated by looking them into properties file. Example with the following Java code:

```
@RequestMapping("${my.home}")
public class HomeController {

    @RequestMapping(value="${my.service1}")
    @ResponseBody
    public String method0(){
        return "method0";
    }

}
```

With properties file:

```
my.home=/home
my.service1=/method0
```

will generate:



SimpleUrlHandlerMapping

The *SimpleUrlHandlerMapping* mapping class allows to specify the mapping of URL patterns to handlers (methods of *Controllers*). This class can be declared using XML bean definitions in at least three different ways as shown below (all of them being supported). Only the servlet XML files explicitly or implictly (default) referenced in the *web.xml* deployment configuration file are considered. For each URL path we will create a different *Spring MVC Any Operation* object.

Method 1

```
<beans ...>
        <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
          <property name="urlMap">
                  <map>
                <entry key="/home.htm">
                  <ref local="homeController"/>
                </entry>
               ...
                 </map>
           </property>
        </bean>

        <bean id="homeController" class="com.castsoftware.example.HomeController" />
</beans>
```

Method 2

```
<beans ...>
        <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
           <property name="mappings">
                <value>
                   /home.htm=homeController
           ...
                </value>
           </property>
        </bean>

        <bean id="homeController" class="com.castsoftware.example.HomeController" />
</beans>
```

Method 3

```
<beans ...>
        <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
           <property name="mappings">
        <props>
            <prop key="home.htm">homeController</prop>
            ...
                </props>
           </property>
        </bean>

        <bean id="homeController" class="com.castsoftware.example.HomeController" />
</beans>
```

In addition we can have further mapping to specific methods of a Controller through the **methodNameResolver** property.

```
<bean id="manageSomeController"
    class="com.cingular.km.lbs.gearsadmin.web.controller.DTVAddressDataController ">
    ...
    <property name="methodNameResolver">
       <bean
         class="org.springframework.web.servlet.mvc.multiaction.PropertiesMethodNameResolver">
         <property name="mappings">
           <props>
             <prop key="/*/deleteAddress">deleteAddress</prop>
           ...
```

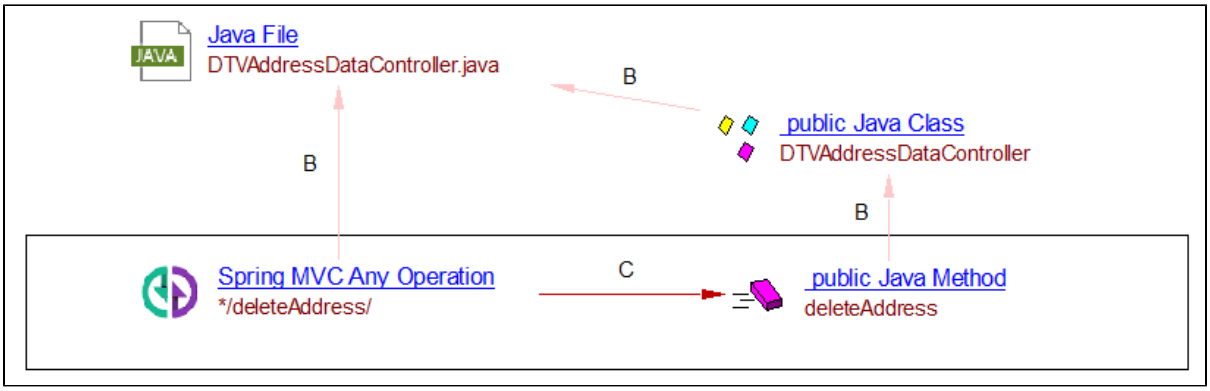
Creation of links between web service operations and controller methods

When defining a URL mapping to a given controller class, each type of controller (determined by its ancestors) can have a set of overridden methods to further customize the response behavior to a URL loading. These methods are usually called by the framework itself (such as *onSubmit* in the *SimpleForm Controller*). The *com.castsoftware.springmvc* analyser will create a callLink from the *Spring MVC Any Operation* object and to each of the overridden methods of the corresponding controller. Potential overriding of default intra-method calls is ignored. When using the XML property PropertiesMethodName Resolver, the **C**allLinks are created to the explicitly referenced callback methods (custom methods added to the controller class). For example, returning to the code snippet above illustrating the use of the *methodNameResolver* property, we can observe a reference to the method of the controller defined below:

DTVAddressDataController.java

```
public class DTVAddressDataController extends MultiActionController {


    public ModelAndView deleteAddress(HttpServletRequest request, HttpServletResponse response) throws
Exception {
                ....
        }
    ...
}
```

The corresponding link would appear as below:



Note that the Spring *MVC Any Operation* will be created below the same Java File (denoted by the **B**elong link) where the referenced controller is found. In addition to the above mentioned links, a ***c**allLink* from the *Spring MVC Any Operation* object to the controller class' constructor method (if present) will be created when using the above mentioned Method 1-2-3 mapping approaches.

Support of BeanNameUrlHandlerMapping, ControllerClassNameHandlerMapping

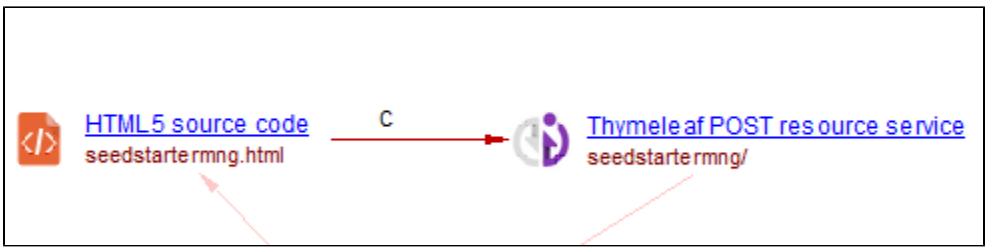
These similarly work by configuring web services via xml files (web.xml, dispatcher-servlet.xml, ...) as SimpleUrlHandlerMapping (see above), but with different rules.

Support of thymeleaf

The following syntaxes are supported for thymeleaf templating:

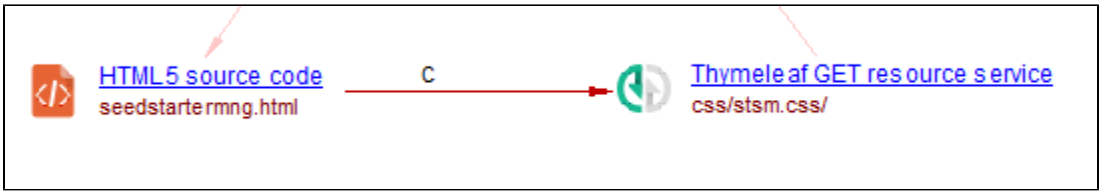
```
      <form ... th:action="@{/seedstartermng}"... method="post">
```

Will create a link from the HTML5 content to a call to a webservice with url '/seedstartermng'



'th:href' are also supported:

```
<... th:href="@{/css/stsm.css}"/>
```



Support for User Input Security

Service entry points are created automatically for applications that have a presentation layer based on SpringMVC with @RequestMapping (and associated annotations) usage. This can be seen in the analysis log file as follows:

```
2018-09-13 10:00:40,148 INFO SecurityAnalyzer.FlawAnalysisEnvironment LoadBlackboxesForApplication cast#spec cast#lib SpringMVCServiceEntryPoints
```

This corresponds to the generation of a file in the following location:

```
<BytecodeFolder>\com.castsoftware.springmvc\ServiceEntryPoints.blackbox.xml
```

> Note that while the ServiceEntryPoints.blackbox.xml file is generated when the extension is used with any release of CAST AIP, it will only be exploited by the CAST User Input Security feature in CAST AIP 8.3.3.

Function Point, Quality and Sizing support

This extension provides the following support:

- **Function Points (transactions)**: a green tick indicates that OMG Function Point counting and Transaction Risk Index are supported
- **Quality and Sizing**: a green tick indicates that CAST can measure size and that a minimum set of Quality Rules exist

Function Points (transactions)	Quality and Sizing
✓	✗

CAST AIP compatibility

This extension is compatible with:

CAST AIP release	Supported
8.3.x	✅
8.2.x	✅
8.1.x	✅
8.0.x	✅
7.3.4 and all higher 7.3.x releases	✅

Supported DBMS servers

This extension is compatible with the following DBMS servers:

CSS	✅
Oracle	✅
Microsoft	❌

Prerequisites

✅	An installation of any compatible release of CAST AIP (see table above)

Dependencies with other extensions

Some CAST extensions require the presence of other CAST extensions in order to function correctly. The **Spring MVC** extension requires that the following other CAST extensions are also installed:

- **Web services linker service** (internal technical extension)

> ⓘ Note that when using the **CAST Extension Downloader** to download the extension and the **Manage Extensions** interface in **CAST Server Manager** to install the extension, any dependent extensions are **automatically** downloaded and installed for you. You do not need to do anything.

Download and installation instructions

Please see:

- Download an extension
- Install an extension

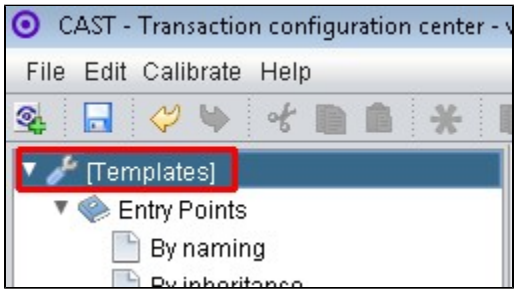
> ⓘ The latest release status of this extension can be seen when downloading it from the CAST Extend server.

CAST Transaction Configuration Center (TCC) Entry Points

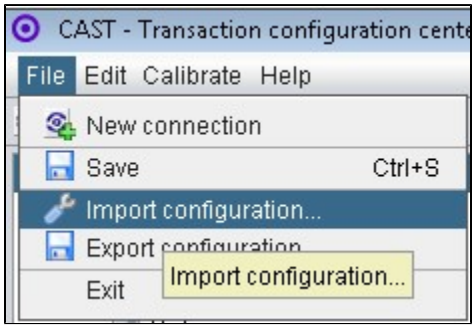
In **Spring MVC 1.3.x**, if you are using the extension with **CAST AIP 8.3.x**, a set of Spring MVC specific **Transaction Entry Points** are now **automatically imported** when the extension is installed. These **Transaction Entry Points** will be available in the CAST Transaction Configuration Center:

Manual import action for CAST AIP 8.2.x

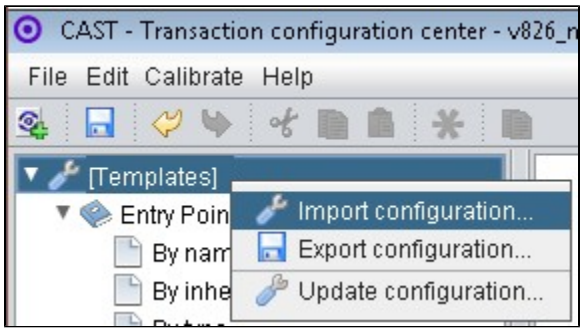
- Locate the .TCCSetup file in the extension folder: **Configuration\TCC\Base_Java_SpringMVC.TCCSetup**
- In the CAST Transaction Configuration Center, ensure you have selected the **Templates** node:



- This .TCCSetup file is to be imported into the CAST Transaction Calibration Center using either the:
  - **File** > **Import Configuration** menu option:

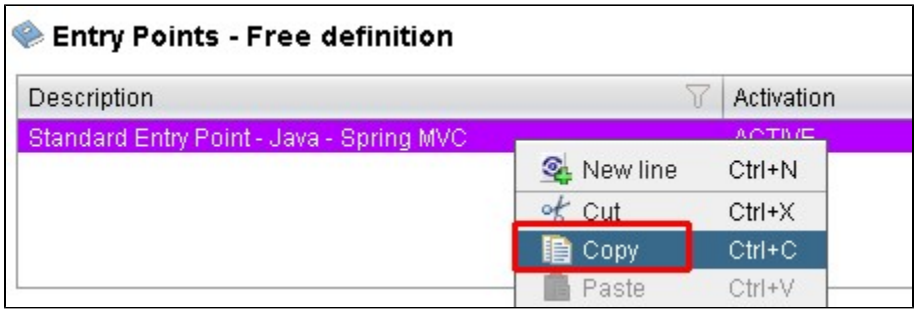


  - Or right clicking on the **Template node** and selecting **Import Configuration**:

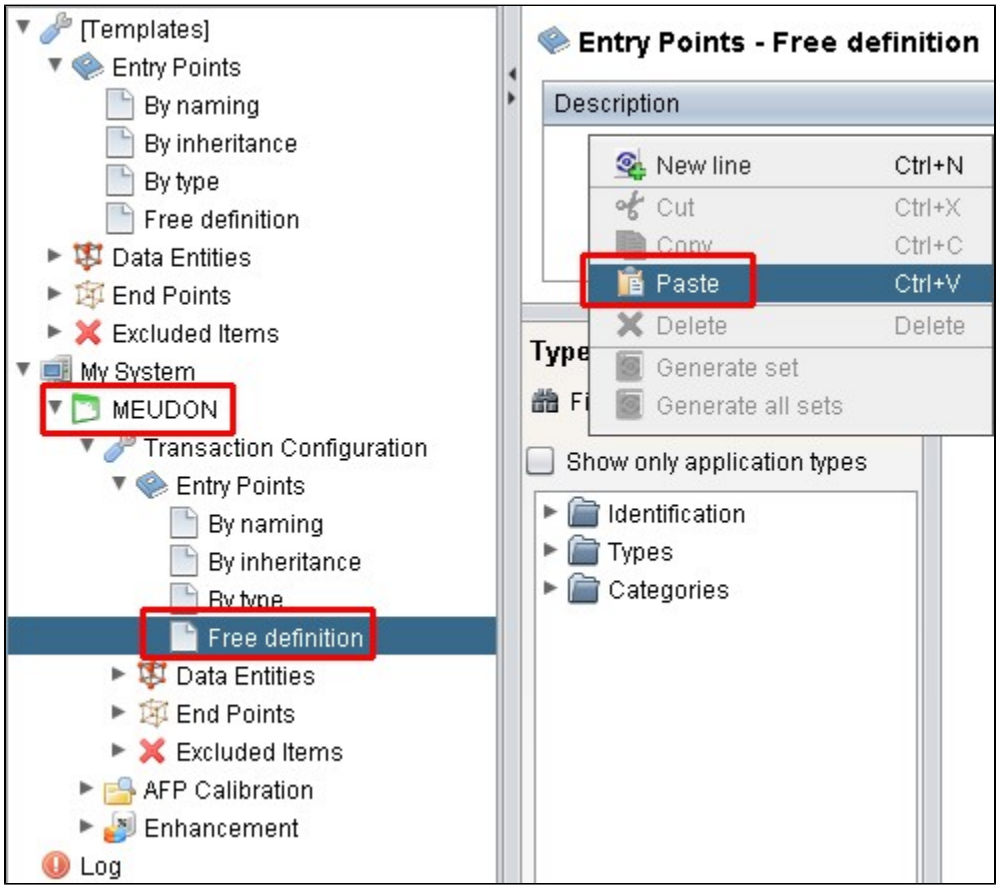


- The import of the "**Base_Java_SpringMVC.TCCSetup**" file will provide you with a sample Transaction Entry point in the **Free Definition** node under **Templates**:

- Now right click the "**Standard Entry Point**" item and select copy:



- Paste the item into the **equivalent node** under the **Application**, for example, below we have copied it into the **Application MEUDON**:



- Repeat for any additional items or generic sets that have been imported from the .TCCSetup file.

Packaging, delivering and analyzing your source code

Once the extension is installed, no further configuration changes are required before you can package your source code and run an analysis. The process of packaging, delivering and analyzing your source code does not change in any way:

- **Package and deliver** your application (that includes source code which uses **Spring MVC**) in the exact same way as you always have.
- **Analyze** your delivered application source code in the CAST Management Studio in the exact same way as you always have - the source code which uses **Spring MVC** will be detected and handled correctly.

Log messages

Warnings

Message ID	Message Type	Logged during	Impact	Remediation	Action

SPMVC-001	Warning	Analysis	Various	None	Contact CAST Technical Support
SPMVC-002	Warning	Analysis	A property could not be found so we could not determine the name of a service.	Provide properties file with the value	

Regular logs

We inform about duplicated entries in servlet configuration files and whether multiple controllers are associated for a single URL path (we don't filter mappings by order priorities).

```
Info : [com.castsoftware.springmvc] Found duplicated entry for path: /otherAddress
Info : [com.castsoftware.springmvc] Found duplicated entry for path: /manageAccount
Info : [com.castsoftware.springmvc] Found multiple controllers for path: /addUser
```

AIA expectations

At the end of the log file you will find a message indicating how many services have been found:

```
5 Spring-MVC web service operations created.
```

Note on JEE analyser warnings

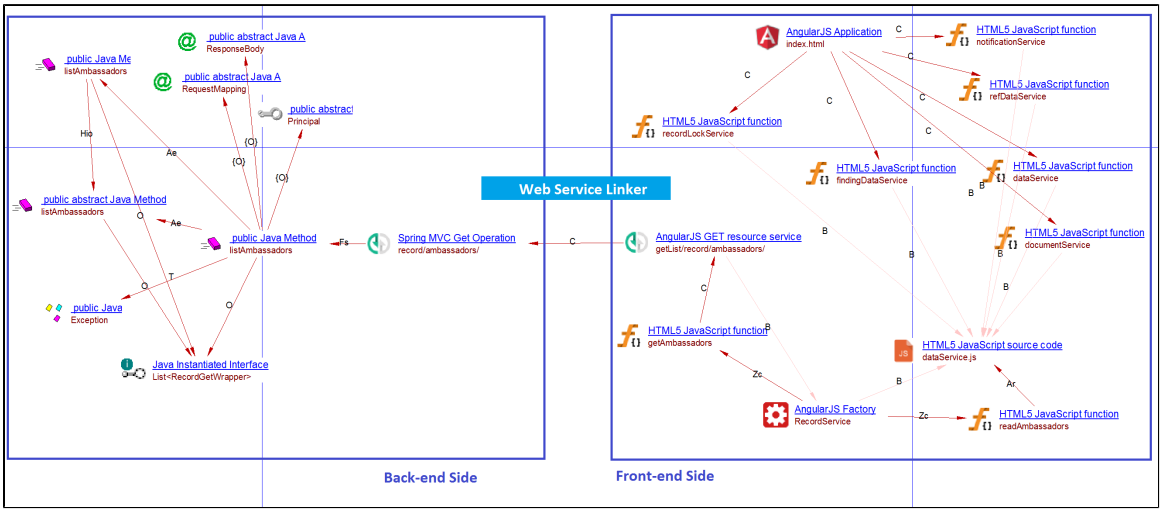
The JEE analyser will complain that some annotations specific to SpringMVC are not handled by an environment profile while they are handled by this extension; you can simply ignore these warnings:

```
Warning MODULMSG ; Job execution    Annotation 'org.springframework.web.bind.annotation.RequestMapping' is not
managed in Environment Profiles
Warning MODULMSG ; Job execution    Annotation 'org.springframework.web.bind.annotation.GetMapping' is not
managed in Environment Profiles
...
```

What results can you expect?

Once the analysis/snapshot generation has completed, **HTTP API** transaction entry points will be available for use when configuring the CAST Transaction Configuration Center. In addition, you can view the results in the normal manner (for example via CAST Enlighten).

Click to enlarge:



Note on image: The links of type *fire* (F) are substituted by *callLink*s (C) in *com.castsoftware.springmvc*versions >=1.5.3.

Objects

Server side

The following objects are displayed in CAST Enlighten on the Java side:

Icon	Description
	Spring MVC Delete Operation Service
	Spring MVC Get Operation Service
	Spring MVC Post Operation Service
	Spring MVC Put Operation Service
	Spring MVC Port
	Spring MVC Service

Rules

None.

Known limitations

- `Order priority in SimpleUrlHandlerMapping is ignored (for each mapping found a web service operation will be created).`
- `The less common programmatic configuration of web services is not supported.`