Annotations

ConfigureMe supports following annotations:

Annotation	Target	Parameters
AbortedConfiguration	Method	
AfterConfiguration	Method	
AfterInitialConfiguration	Method	
AfterReConfiguration	Method	
BeforeConfiguration	Method	
BeforeInitialConfiguration	Method	
BeforeReConfiguration	Method	
Configure	Field	
ConfigureMe	Class	name,watch,type, allfields
DontConfigure	Field	
Set	Method	value
SetAll	Method	
SetIf	Method	value,condition

Details

AbortedConfiguration

Called if the configuration has been aborted due to a casting or parsing error, for example trying to set an int field with a unparseble string value.
Example:

```
@AbortedConfiguration public void callIfAborted(){
        System.out.println("Configuration has been aborted");
}
```

AfterConfiguration

Called after each successful configuration.
Example:

```
@AfterConfiguration public void callAfterEachConfiguration(){
        System.out.println(this+" has been configured");
}
```

AfterInitialConfiguration

Called after **first** successful configuration.
Example:

```
@AfterInitialConfiguration public void callAfterInitialConfigurationOnly(){
        System.out.println(this+" has been INITIALY-configured");
}
```

AfterReConfiguration

Called after each successful re-configuration.
Example:

```
@AfterReConfiguration public void callAfterReConfigurationOnly(){
        System.out.println(this+" has been RE-configured");
}
```

BeforeConfiguration

Called before each configuration attempt.
Example:

```
@BeforeConfiguration public void callBeforeEachConfiguration(){
        System.out.println(this+" will be configured now");
}
```

BeforeInitialConfiguration

Called before **first** configuration attempt.
Example:

```
@BeforeInitialConfiguration public void callBeforeInitialConfigurationOnly(){
        System.out.println(this+" will be INITIALY configured now");
}
```

BeforeReConfiguration

Called before each configuration attempt except first.
Example:

```
@BeforeReConfiguration public void callBeforeReConfigurationOnly(){
        System.out.println(this+" will be RE-configured now");
}
```

Configure

Marks a field configureable. The field must be either public or have a setter (recommended).
Example:

```
	@Configure private String greeting;

	public void setGreeting(String greeting) {
		this.greeting = greeting;
	}
```

```
	@Configure public String greeting;
```

ConfigureMe

Marks a class as configurable.

Parameter	default	Description
name		Configuration name. If skipped the class name (without package) is used.
watch	true	If true the configuration for the artefact will be watched and the artefact reconfigured as soon as the config changes. It implicitely means that the instance to the artefact will be stored in the configuration management. Don't use on objects which are supposed to die soon after usage (at the end of a request or similar cause it could lead to memory leaks.
type	Type.File	Type of configuration source. Currently supported are File and Fixture.
allfields	false	If true all fields are set from configuration regardless whether they are annotated *Configure* or not.
		Only fields which are marked *DontConfigure* are ignored.

Example:

```
@ConfigureMe(name="fixture", type=ConfigurationSourceKey.Type.FIXTURE, watch=false)
public class TestConfigurable {
```

```
@ConfigureMe(allfields=true)
public class HelloWorld {
```

DontConfigure

Marks a field not configureable. In case the class is annotated with ConfigureMe(allfields=true) this field will be ignored.
Example:

```
	@DontConfigure private String myField;
```

Set

Calls the method with the value of the configuration property specified by the value (*noname*) attribute.

Parameter	default	Description
value		Name of the attribute in the configuration object

Example:

```
@ConfigureMe(name="helloworld", watch=false)
public class LanguageResearcher {
        @Set("greeting")
        public void research(String greeting){
                System.out.println("\tI found out that here people are greeting with \""+greeting+"\"");
        }
}
```

SetAll

Calls the method with each name, value pair in the configuration object. Requires the target method to have two string parameters.
Example:

```
@SetAll
public void debug(String name, String value){
        System.out.println(""+name+" is configured as "+value);
}
```

SetIf

Calls the method with the name and the value of the configuration properties which match value and condition annotation parameters. Value is a string parameter, while condition is one of the SetIfCondition enum values. There are currently 3 of them :

- startsWith (does the key start with given annotation value)
- contains (does the key contain given annotation value)
- matches (does the key match given annotation value)
  All of conditions are checked by calling the String methods of the same name on attribute name (so, the last condition supports regular expressions).

Parameter	default	Description
value		Pattern of the attribute name in the configuration object
condition	SetIfCondition.matches	Condition, which regulates the configuration properties to be passed to the annotated method

Example:

```
@ConfigureMe(name="serverRegistry", watch=false)
public class ServerRegistry {
        @SetIf(value="server", condition=SetIfCondition.startsWith)
        public void addServer(String serverName, String serverUrl){
                System.out.println("\tAdded server to registry - \""+serverName +" : "+serverUrl+"\"");
        }
}
```

While processing SetIf annotation in the previous peace of code, the method will be called with all of the configuration properties, which have names starting with "server".