

eHealth Framework

Validating a Service via Annotations

Notice

The wording in this document applies equally to women and men. The masculine form was selected to ease the comprehensibility and legibility of the text.

All company logos are a registered trademark of InterComponentWare AG.

The product names mentioned in this documentation are either trademarks or registered trademarks of the respective owners and are stated for identification purposes only.

This documentation and the software components are protected by copyright © 2006-2009 InterComponentWare AG.



Note:

The current version of this document has a draft status and various chapters are still in review.

The document is collaboratively built with the use of the Darwin-Information-Typing-Architecture (DITA) and has therefore a draft status concerning styles and layout. The necessary adaptations are currently also in a developmental stage.

All rights reserved.

Contents

1 Overview	1
2 Input Validation via Annotations	2
3 Validation Enforcement	3
4 Writing Your Own Validator	5
5 Configuration	6

1 Overview

Purpose

The howto explains how input validation can be switched on for classes and methods, using specific annotations of eHF.

Scope

You will see how you must annotate classes and methods in order to validate input. Further the necessary Spring configuration will be explained and last but not least you will get a short guidance to implement your own validator.

2 Input Validation via Annotations

Input validation is the key to “Defensive Programming”. Checking input of a method and to reject it in a controlled fashion will minimize the risk to run into unpredictable errors caused by malformed input data.

A “robust” application verifies input data at designated points in the control flow. As a “Rule of Thumb” an application should validate all input provided by external components. What an external component defines depends on the application, it could be:

- a jar file
- a different package
- a different class

3 Validation Enforcement

The eHF has a built in support for input validation based on JAVA 5 annotations:

- `com.icw.ehf.commons.metadata.validator.annotation.Validate`
- `com.icw.ehf.commons.metadata.validator.annotation.Param`
- `com.icw.ehf.commons.metadata.validator.annotation.SuppressValidation`

The input validation is enforced by intercepting a method call and if the `@Validate` annotation is found, the input parameter are validated against a pre-configured validator. The following code snippet shows how the annotations are used:

```

1  @Validate
2  public class ValidatingService implements ServiceInterface {
3
4  @SuppressValidation
5  public String serviceCall_I (String str) {...}
6
7  public String serviceCall_II (@Validate(
8      validator = "StringValidator",
9      params = {
10     @Param(key="KEY1", value="VALUE1"),
11     @Param(key="KEY2", value="VALUE2")
12     }
13 ) String token) {...}
14
15 public Object serviceCall_III (Object token) {...}
16 }

```

In line 1 the class is annotated with `@Validate`. This is necessary to enable input validation. Now all input of all methods will be validated. In line 4 the method `serviceCall_I` is explicitly marked not to validate any input by the annotation `@SuppressValidation`. The class wide input validation can be overwritten by using the `@Validate` annotation shown in lines 7- 12. The annotation variable `validator` is interpreted internally, e.g. a `java.lang.Class` can be used, or a Spring bean name.

The algorithm that fetches the `@Validate` annotation is described below in a “pseudo code” style:

```

METHOD build(Method method, Class baseClass) {
  IF (baseClass is annotated with Validate) AND
  IF (baseClass declares the method) THEN
  {
    IF (method is annotated with SuppressValidation) THEN
    {
      nothing to do RETURN
    }
    ELSE {
      IF NOT (Validate annotation support propagation) THEN
      {
        remember method and class
        RETURN
      }
    }
  }
  IF (get the superclass) AND
  IF NOT (superclass IS NOT java.lang.Object) THEN
  {
    CALL build(method, superclass)
  }
}

```

Validating a Service via Annotations

```
FOREACH (interFace of baseClass) DO
{
    CALL build(method, interFace)
}
RETURN
}
```

The input validation integrates with the Spring framework. The implementation provides adaptors for ordinary Spring AOP. This means that the annotated classes must be registered in the Spring context (see [Configuration](#) on page 6).

4 Writing Your Own Validator

Implementing your own validator is quite simple. You just have to provide an implementation of the interface `com.icw.ehf.commons.metadata.validator.Validator`. To use this validator in the context of *eHF*, it is necessary to supply an instance of this validator in the *Spring* configuration file (see [Configuration](#) on page 6).

5 Configuration

As already mentioned in [Validation Enforcement](#) on page 3 validation is enforced by intercepting a method call. The “Aspect Oriented Programming (AOP) Paradigm” is especially useful to introduce logic into business code which is not directly related to use cases. eHF provides a Spring integration to intercept method calls to “Spring Beans”. Input validation can be either enforced using the classical Spring AOP mechanisms, or an eHF proprietary way.

```

1  <bean class="org.springframework.aop.
2      framework.autoproxy.DefaultAdvisorAutoProxyCreator"/>
3
4  <bean id="validatorRegistry" class=
5      "com.icw.ehf.commons.metadata.validator.ValidatorRegistryImpl">
6      <property name="validatorRegistry">
7          <list>
8              <bean class="com.icw.ehf.commons.
9                  metadata.validator.adapter.StringMetaDataValidatorAdapter"/>
10
11                 <bean class="com.icw.ehf.commons.metadata
12                     validator.adapter.IntegerMetaDataValidatorAdapter"/>
13             </list>
14         </property>
15     </bean>
16
17 <bean id="descriptorCache" class="com.icw.ehf.commons.metadata.
18     validator.method.descriptor.MethodDescriptorCache">
19     <property name="validatorRegistry">
20         <ref bean="validatorRegistry"/>
21     </property>
22 </bean>
23
24 <bean id="inputValidationPointcut" class="com.icw.ehf.commons.
25     metadata.validator.aop.spring.MethodValidationPointcut">
26     <property name="descriptorRepository">
27         <ref bean="descriptorCache"/>
28     </property>
29 </bean>
30
31 <bean id="inputValidationInterceptor" class="com.icw.ehf.commons.
32     metadata.validator.aop.spring.MethodValidationInterceptor">
33     <property name="descriptorRepository">
34         <ref bean="descriptorCache"/>
35     </property>
36 </bean>
37
38 <bean id="inputValidationAdvice" class="com.icw.ehf.commons.metadata
39     validator.aop.spring.MethodValidationAdvice">
40     <constructor-arg>
41         <ref bean="inputValidationPointcut"/>
42     </constructor-arg>
43     <constructor-arg>
44         <ref bean="inputValidationInterceptor"/>
45     </constructor-arg>
46 </bean>

```

In the lines **4-15** the validator registry is configured. This registry is used to resolve the validators which are included in the `@Validate` annotation. The resolution of validators is either explicitly by a concrete name, or implicitly by the class of the parameter of the validated method. How the `validator` field of the `@Validate` annotation is interpreted depends on the implementation of the registry (here it's the class name). Lines **17- 22** are

a cache which speed up the retrieval of descriptors for a method. The lines **24- 46** enable the Spring AOP method interception.

In the context of eHF, you can use a *ServiceStrategy* (see “eHF Reference Documentation” ; Sec. “Service Strategies”) to enforce input validation. The following sample configuration declares the Spring Bean *validatorServiceStrategy* in line **11**, which implements the *ServiceStrategy* interface.

```
1 <bean id="validatorRegistry" class=
2     "com.icw.ehf.commons.metadata.validator.ValidatorRegistryImpl">
3     [...]
4 </bean>
5
6 <bean id="descriptorCache" class="com.icw.ehf.commons.metadata.
7     validator.method.descriptor.MethodDescriptorCache">
8     [...]
9 </bean>
10
11 <bean id="validatorServiceStrategy" class="com.icw.ehf.commons.
12     metadata.validator.aop.spring.MethodValidationServiceStrategy">
13     <property="descriptorRepository">
14         descriptorCache
15     </property=" ">
16 </bean>
```

In the configuration file above, the code between the lines **1-9** configures the descriptors and the descriptor repository. The bean in **11** is the implementation of the *ServiceStrategy*. This bean must be injected into the module specific `com.icw.ehf.commons.aop.ServiceStrategyInterceptor` (see “eHF Reference Documentation” ; Sec. “Service Strategies”).