ESFINGE METADATA FRAMEWORK

INTRODUCTION

The Esfinge Metadata framework is a framework for metadata retrieving and validation, aiming to ease and assist the development of frameworks that uses annotations. The API provides methods for reading and retrieving metadata at runtime. It has a API for individual metadata reading and a mapping API for retrieving the metadata to a "MetadataContainer" class, which is responsible for storing metadata at runtime. To simplify the explanation, only the annotations, and classes used on the experiment will be presented in this document. Please take your time to read attentively to this documentation.

METADATA READING ANNOTATIONS

The Esfinge Metadata API provides annotations for reading metadata from elements. Normally the annotations are read from the element and stored in a single class that contains these informations. In the Esfinge Metadata API this class is called MetadataContainer. This feature maps the annotations for one class, facilitating the work of metadata reading. Following are the annotations used for reading the metadata from a class.

@ContainerFor(Enum ContainerTarget) - Receives as parameter a value from the ContainerTarget Enum, for instance if the value is ContainerTarget.TYPE, then the container will store metadata for a element of type class. Other allowed type are fields and methods.

@ElementName - This annotation can be used on fields, methods or classes. It maps the element name for the metadata container.

@ReflectionReference - This annotation maps the annotated element type to the metadata container. It can be used in classes, fields, or methods.

@AnnotationProperty(Class annotationClass,String property) - This annotation verifies if there is a annotation from "annotationClass" class and check if exist the property with name "property".This annotation is used in fields and methods.

@ContainsAnnotation(Class value) - This annotation verifies if the the annotation from the "value" class exists on the target element. This annotation must be used on fields of type boolean.

@ProcessFields - This annotation process the fields information and places it in a information container. This annotation must be used in objects of type List and Set.

EXAMPLE OF USAGE

```
@ContainerFor(ContainerTarget.TYPE)
public class ContainerClass {

   @ElementName
   private String className;

   @ReflectionReference
   private Class<?>  reference;

   @ProcessFields
   private List<FieldContainer> fieldContainer;

   @ContainsAnnotation(Table.class)
   private boolean table;

   @AnnotationProperty(annotation = Table.class,
                       property ="name")
   private String tableName;


//GETTERS AND SETTERS ARE OMITTED

}
```

The class CointainerClass will store information for a class element. Since it is annotated with @ContainerFor(ContainerTarget.TYPE)

The field className of the class ContainerClass will receive the annotated element name.

The field reference of the class ContainerClass will receive the reflection reference of the annotated field.

The List fieldContainer will contain all information about the processed fields of the annotated element.

The boolean field table will be true if the element is annotated with @Table, else it will be false.

The String tableName will receive the value of the property name of the annotation @Table if it is present in the element.

ESFINGE METADATA FRAMEWORK

CODE CONVENTIONS - DEFINING ANNOTATIONS EXISTENCE

The Esfinge Metadata API provides a way to define code conventions that can be used instead of the annotation. That means that, if an element has that code convention, it will be the same as if it has the annotation. To do that you need to add a convention annotations to the target annotation.

For instance, if we want that the annotation @Configurable have a convention for methods started with “config”. That means that for Esfinge Metadata API, any method started with “config” is equivalent than having the @Configurable annotation.

Consider the @Configurable annotation definition, and the class **ConfigurableExampleClass**.

```
//annotation definition
@Target(ElementType.METHOD, ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@PrefixConvention(value = “config”)
@HaveAnnotationOnElementConvention(annotationClass = Column.class)
public @interface Configurable{
}

//class definition
public class ConfigurableExampleClass {
   @Column
   private String name;
   @Configurable
   private int value;

   public void configValue(int value){
      this.value = value;
   }

   @Configurable
   public void annotatedMethodForConfigName(String name){
      this.name = name;
   }
}
```

The Esfinge API will consider all methods whose name starts with “config” as having the @Configurable annotation. These type of conventions are are used for considering the existence of the annotation.

The API provides many annotations like these, below we present some other examples.

@SuffixConvention(String value) - This convention specifies if the annotated element name ends with the suffix value.
Example: @SuffixConvention(value = “Test”) defines a convention for all attributes that ends with “Test”.

@FieldTypeConvention(Class type, boolean canBeSubtype) - This convention applies when the element is of the class specified on the parameter **type**. The boolean parameter canBeSubtype is optional, and specifies if the convention also applies to the subtypes of the class **type**.

@HaveAnnotationOnElementConvention(Class annotationClass) - This convention determines if the element has a specific annotation. In this example, if the code element is annotated with @Column, then the framework will consider that it is also annotated with @Configurable.

Both methods are considered by the framework as having the annotation @Configurable, since the method **configValue** has the *prefix convention* and the method **annotatedMethodForConfigName** has the annotation @Configurable.

Both fields are considered by the framework as having the annotation @Configurable, since the field **value** is annotated with @Configurable, and the field **name** follows the *have annotation convention*, because it is annotated with @Column.

***NOTE:**

A annotation can be annotated with as many conventions as you need, however not all of them must apply for the framework to consider the element as having the desired annotation.

ESFINGE METADATA FRAMEWORK

CODE CONVENTIONS - SETTING ANNOTATION ATTRIBUTES VALUES

If the annotation have attributes, you can annotate them to define which values they will have in case the convention is identified. It is fundamental to set values for annotations that are normally used with any parameter. For instance, the javax.persistence @Column annotation can have the parameter name, which allows one to use @Column(name = "myColumn"). Consider the new declaration of the @Configurable annotation having the parameters, **age()** and **name(),** and the new declaration of the **ConfigurableClass**.

```
//Configurable annotation definition
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@PrefixConvention(value = "config")
public @interface Configurable{

  @FixedIntegerValue(value = 21)
  int age();
  @FixedStringValue(value = "bob")
  String name();

}

//class definition
public class ConfigurableExampleClass {

   /* this instance of the class person will have age = 21 and name = bob,
    since those are the values defined with the @FixedIntegerValue and
    @FixedStringValue annotations and the field configPerson has the prefix
    config, specified by the convention.*/
  private Person configPerson;

  /* this instance of the Person class, will have age = 23 and name = alice, since
  the annotation @Configurable has these parameters passed as arguments.*/
  @Configurable(age = 23, name = "alice")
  private Person anotherPerson;

  /*no values will be set for this instance, since it doesn't have the
  @Configurable annotation or the @NameAfterPrefix, and does not have the
  prefix specified by any of the conventions.*/
  private Person randomPerson;

   /*this instance will have name = Person, since it follow the convention
    specified by the annotation @NameAfterPrefix*/
    private Person prefixPerson;
  //GETTERS AND SETTERS ARE OMITTED
  }
  //NameAfterPrefix annotation definition
  @Target(ElementType.FIELD)
  @Retention(RetentionPolicy.RUNTIME)
  @PrefixConvention(value = "prefix")
  public @interface NameAfterPrefix{
    @ElementNameAfterPrefix(prefix = "prefix")
    String name();
  }
```

The @PrefixConvention will specify the existence of the annotation @Configurable annotation to all **field** elements whose name starts with config.

However, this convention will not specify the values of the attributes **age()** and **name()** of the @Configurable annotation. For setting these values *annotations for setting attributes* are needed.

The Esfinge Metadata API provides some annotations for setting all the *primitive types* and the *numeric wrapper types*, such as Float.class and Integer.class.

To set the values of the attributes **name()** and **age()**, you need to use @FixedStringValue and @FixedIntegerValue, respectively, where.

@FixedStringValue(String value) - Will Set the @Configurable annotation attribute **name()** with the string **value**.

and,

@FixedIntegerValue(int value) - Will Set the @Configurable annotation attribute **age()** with the int **value**.

The Esfinge Metadata API provides annotations for setting attribute values based on the element *name*.

@ElementNameAfterPrefix (String prefix) - Will set the @NameAfterPrefix **name()** value with the value of the string **prefix**.

Other possible annotation is @ElementName, which sets the value of the annotated attribute to the annotated element. For instance, if the attribute **name()** of the @NameAfterPrefix was annotated with @ElementName, the the instance prefixPerson of the class **Person** would have the attribute **name** = "prefixPerson".

The API also provides the annotation @ElementNameBeforeSuffix(String suffix). This sets the annotated attribute with the value before a specific **suffix**.