Kermeta Emitter Template

Reference manual

Cyril Faucher
Mickael Clavreul

Abstract

This manual presents the Kermeta Emitter Template language. This is the reference manual for anybody who want to use KET.

Published Build date: 3-November-2010
Published $Date:: 2010-05-06 16:04:29#$

CHAPTER

Preface

Kermeta Emitter Template is a scripting language dedicated to text generation. Similarly to other scripting languages, it offers end-users a simple language that supports interactions with the Kermeta Language to generate text from complex computations.

This document gives beginners an introduction to the KET language, then it offers an overview of the definition of the generator parameters. Follows a definition of main tags, the compilation of KET files to Kermeta files and how to use them to generate text.

Important

KET is part of an evolving software and despite that we put a lot of attention to this document, it may contain errors (more likely in the code samples). If you find any error or have some information that improves this document, please send it to us using the bug tracker in the forge: **http://gforge.inria.fr/tracker/?group_id=32** or using the developer mailing list (kermeta-developers@lists.gforge.inria.fr) Last check: v1.2.0

Tip

The most update version of this document is available on line from http://www.kermeta.org .

CHAPTER 1

Introduction to KET

KET is based on the JET [1] [2] engine enhanced by the syntax of Kermeta for the model navigation. It is also very similar to Velocity or JSP. This is a user-friendly text-based template engine wich generate text from a KET template and a model. Templates are composed of two main elements: text that should be written in the output and tags that are interpreted to generated string values from some computation. Once templates are ready to be compiled, they are interpreted by the template engine. It creates a new Kermeta file that declares a generator as a generate(params) method.

Warning

Using KET is efficient when templates contains more text that expressions (i.e. a text with holes). If you need more computation than text, you should consider writing a pretty printer directly in Kermeta [3] .

Tip

Since everything goes down to a kermeta code, you can also combine the approaches KET/kermeta pretty printer

[1] http://www.eclipse.org/modeling/m2t/?project=jet#jet
[2] http://help.eclipse.org/ganymede/index.jsp?nav=/27
[3] Writing a pretty printer is a kind of visitor which is quite ease to write thanks to kermeta aspects.

CHAPTER 2

Generator Parameters

A KET template starts with the definition of the tag `<%@ket %>`. This tag is helpful to define how the Kermeta file should be generated. There is eight parameters either mandatory (m) or optional (o):

- **package (m)** - root package of the Kermeta file
- **require (m)** - set of requires that the Kermeta file should contain (separated by whitespaces)
- **using (m)** - set of using that the Kermeta file should contain (separated by whitespaces)
- **class (m)** - main class of the Kermeta file
- **isAspectClass (o)** - true if the main class should be an aspect (reopen an existing class)
- **operation (o)** - name of the main operation
- **isMethod (o)** - true if the main operation is a redefinition of an existing operation
- **parameters (m)** - parameters of the `generate(...)` method. Those parameters can be used in KET tags to acquire data from outside the generator

```
<%@ket
package="example"
require="my_require other_require"
using="package1 package2"
isAspectClass="false"
class="example"
isMethod="false"
operation="generate"
parameters=""
%>
```

CHAPTER 3

Tags and Syntax

KET provides four types of tags.

3.1. Comments

KET templates may contain comments. Comments are defined between the characters <%-- and --%>. Comments have no impact on the execution of the template, except that they may influence whitespace stripping rules. KET comments are copied to the generated kermeta class as Kermeta comments. KET templates accept two special tags in the first non-blank line of a comment. The tag `@header` will cause the comment to be emitted as the file header comment for the generated Kermeta class. This is generally useful to insert copyright notices into the generated Kermeta code. The tag `@class` will cause the comment to be emitted as the class Kermeta doc comment for the generated Kermeta class.

3.1.1. Valid comments

Comments may span several lines, and may contain any text.

```
<%-- @header
This comment will appear as the file header comment in the generated kermeta code
--%>
<%-- @class
This comment will appear as the kermeta class doc comment in the generated kermeta code --%>
<%-- This comment will not appear in the template output --%>
<%-- This directive is not used <%= attr.name%> --%>
```

3.1.2. Invalid comments

Comments may not appear within other KET elements.

```
<%= attr.name <%-- illegal comment --%> %>
```

3.1.3. Escaping comments characters

To emit the characters <%-- in a template output, enter <\%--. To emit --%>, enter --%\>.

```
<\%-- this will show in the template
```

```
                output --%\>
```

3.2. Kermeta expressions

KET templates may emit the result of a Kermeta expression by enclosing the Kermeta expression between the characters <%= and %>. One scenario is to compute some data in Kermeta and include the result into your template .

```
The name of the class executing is:
    <%= aClass.name %>
This is the <%= 5th %> execution of the generator: it says<%= "hello" %>.
```

3.2.1. Valid expressions

Expressions contain valid Kermeta expressions. Expressions may access any Kermeta element in scope, including generator parameters or field and methods declared in Kermeta scriptlets. The emitted Kermeta code for the template will evaluate the Kermeta expression and convert the result to a String (if necessary). Expressions may be constants but it looks weird.

```
<%= 3 + 4; %> <%-- semicolon not allowed in
                Kermeta expressions --%>
```

3.2.2. Invalid expressions

Expressions are not statically checked to any error in the Kermeta expression will only be detected in the emitted Kermeta code. Errors are not correlate back to the KET template.

3.2.3. Escaping expressions characters

To emit the characters <%= in a template output, enter <\%=. To emit %>, enter %\>.

3.3. Kermeta scriplets

KET templates may contain sections that contain Kermeta statements by enclosing the Kermeta expressions between the characters <% and %>.

3.3.1. Valid scriptlets

Scripts may contain one more more valid Kermeta statements or blocks. A scriptlet may also include a partial Kermeta block, so long as a subsequent scriptlet completes it. Scriplets may reference any Kermeta elements in scope, including variables declared in other scriptlets, and methods and fields declared in Kermeta declara-

tions. The emitted Kermeta code from the template will contain the Kermeta statements in the generation method.

```
<% var x : Integer init 2 %>
<% var y : Integer init x*5 %>
<% if y >= 10 then %>
  Y is >= 10
<% end %>
```

3.3.2. Invalid scriptlets

Expressions are not statically checked to any error in the Kermeta expression will only be detected in the emitted Kermeta code. Errors are not correlate back to the KET template.

3.3.3. Escaping scriptlets characters

To emit the characters <% in a templates output, enter <\%. To emit %>, enter %\>.

CHAPTER 4

Compilation with Kermeta

Ket templates should be compiled to generate a Kermeta file. Right click on your KET file and select *Kermeta -> Translate Kermeta Template*. You get a new .kmt file with the same name of your KET template. The .kmt file contains a Kermeta class that conforms to the parameters you provide. This Kermeta class provides also a `generate(...)` operation you would execute to pretty-print the text you expect.

```
<%@ket
package="hello_world"
require=""
using=""
class="Hello_World"
parameters=""
%%>
<%-- Pretty printer code --%>
Hello World !!
```

```
package hello_world;
require kermeta
using kermeta::standard
using kermeta::utils
class Hello_World{
operation generate():String is do
var _res: StringBuffer init
StringBuffer.new

// Pretty printer code
_res.append("Hello World !!")
result := _res.toString
end
}
```

4.1. Nesting templates

If you need to nest templates because you produce modular templates that can be included in each other, one way is to enclose the `generate(...)` operation you get from the generated Kermeta file into a Kermeta scriptlet.

CHAPTER 5

How to use generated code in Kermeta programs

Once your templates have been properly compiled into Kermeta files, you can call the `generate(...)` method into your main Kermeta code to generate the text.

```
package root_package;
require kermeta
require "templates/generator.kmt"
using kermeta::standard
using kermeta::utils
using hello_world
class Main{
operation main() : Void is do
	// Generator initialization
	var gen : Hello_World init Hello_World.new

	// Displays text generated on the console output
	stdio.writeln(gen.generate())
end
}
```

CHAPTER 6

Online resources

You'll find additional resources and samples online : http://www.kermeta.org/mdk/ket.

Additionally, you can post questions on Kermeta mailing lists (http://gforge.inria.fr/mail/?group_id=32) and forums (http://gforge.inria.fr/forum/?group_id=32).