MODULE REFERENCE MANUAL

Andrea Carboni, Misko Hevery

Rev 4, 26-Apr-2004

Contents

Chapter 1

The Velocity module

1.1 Introduction

The velocity template engine (http://jakarta.apache.org/velocity/index.html) is a tool from the jakarta project used to create ascii files from templates. Giving access to the druid core, velocity can generate any kind of ascii file: summaries, docs, code in plain ascii or other ascii based format (like XML). First, you decide what you want to generate, then it is only a matter of writing the template.

1.2 The module

The velocity module is located at the database level, Generation tab. Select the Modules subtab and do a right-click on the From templates node. Choose Add module and then Velocity from the popup. Now the module has been added to your project. You should see something like figure 1.1.

In the Output Directory textfield you can specify the directory where velocity will generate the files. If you leave this option empty it will not be used but you have to provide a valid path in the Output table column.

In the Options tab you can see options specific to the velocity module. Each table row represents a *generator* (see section 1.3). When you select a row, in the Description area you can

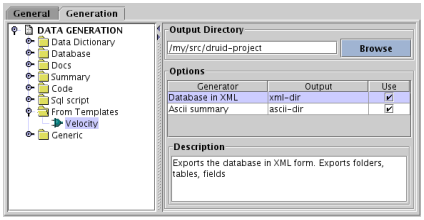

Figure 1.1: The velocity module

see what the generator does. For each generator you can set the following options:

- **Output**

  This is another output directory that can be used together or not with the Output Directory option. If you specify both, the final path would be:

  ```
  <output-directory>/<output>/<generated-data>
  ```

  If you specify only output the final path would be :

  ```
  <output>/<generated-data>
  ```

  If Output Directory is not blank but output starts with a slash then the same previous path will be used (output would be treated as an absolute path).

- **Use**

  If checked, the velocity module will activate the corresponding generator.

1.3 Generators

A generator is a set of commands that drive the generation process. Each generator can generate one or more files, depending on what you want to generate (a single summary file or several code files). Generators are located into the `<druid-dir>/data/templates/velocity` directory. Each directory you see here contains a generator and must contain at least the "generator.xml" file. This file contains a description of the generator and the commands that must be issued. A typical generator.xml file is like this:

```
<?xml version="1.0"?>
<generator>
   <properties>
      <descr>...</descr>
      <longDescr>...</longDescr>
   </properties>
   <commands> ... </commands>
</generator>
```

The properties tag contains only two tags:

- **descr**

  This is a simple description that compares in the table, like "Ascii summary"

- **longDescr**

  This is a general description of what the generator does and is shown at the bottom of the panel

The commands tag contains the commands that will be issued when the generator is activated. You can specify one or more of:

- **apply**

  apply is used to apply a template to the database. Syntax example:

  ```
  <apply template="template.vm" output="database.xml"/>
  ```

  template.vm is the template filename that must exist into the same generator directory.

  output is the output file name to generate. If it starts with "/" it is considered an absolute path.

- **copy**

  copies a file from the druid directory (o something else) into the generation directory. Useful if you want to copy images for generated html. Syntax example:

  ```
  <copy file="images/image.gif" to="images/image.gif"/>
  ```

  file is the file to copy. If it doesn't start with "/" the path is taken relative to the druid directory

  to is the destination. If it doesn't start with "/" the path is taken relative to the generator output directory

- **loop**

  loop is used to apply the same template to different files. It can be used to generate, for example, java source files because the same template is applied to different objects to produce different files. Syntax example:

  ```
  <loop template="java-template.vm" output="java/$.java" on="table"/>
  ```

  The module will create a collection of tables and pass them to velocity. Each time the module substitute the wildcard '$' with the current object's name. The "on" parameter must be a valid object name, like table, folder, view, procedure ...

- **makeDir**

  just creates a directory, the path is relative to the generation path. Syntax example:

  ```
  <makeDir name="images"/>
  ```

name can be relative to the generator output directory or an absolute path.

The optional on attribute can be used to create several directories and works the same way as in the loop command. When the on attribute is specified, the wildcard '$' can be used in the name attribute and it will be replaced with the current object's name. Example:

```
<makeDir name="$" on="table" />
<loop template="table2class.vm" output="$/$.class" on="table" />
```

1.4 Adding generators

To add a generator follow these steps:

- Create the generator directory inside the data/templates/velocity directory
- Create the generator.xml file into this directory
- Create one or more templates into this directory
- Restart Druid

If you are using an environment which allows different users, make sure that the files have a read permission.

1.5 Writing templates

In order to write a template, you need to know what objects you can access. See the programming guide document for the velocity bindings.

1.6 An example

This is the template for the ascii summary generator. As you can see, it is very simple:

```
#foreach($table in $database.allTables)
================================================
===
=== $table.name
===
================================================

Description

$table.descr

#foreach($field in $table.fields)
```

```
    ------------------------------------------------------
    --- $field.name
    ------------------------------------------------------

    Type          : $field.type
    SqlType       : $field.sqlType
    IsForeignKey  : $field.isForeignKey
    IsPrimaryKey  : $field.isPrimaryKey

    Description

    $field.descr

    #end
    #end
```

Chapter 2

The Castor module

2.1 Introduction

Castor (HTTP://CASTOR.EXOLAB.ORG) is an open source data binding framework for java. The Castor generation module can create the Castor's XML mapping file between the Druid's database schema and the java classes of your application. These classes can also be generated by Druid using the java code generation module.

2.2 The module

The module is located under the node Generic and is named Castor Mapping. There are options at database, table and field level. Figure 2.1 shows some options at database level.

At database level you have the following options:

- **Package** : this is used as a prefix for class names, inside the mapping file.
- **Class suffix** : this is used as a suffix for class names.
- **Includes** : simple strings that will go into the include elements.
- **Key gens** : this tab contains options for key generators that accept parameters (tipically the High/Low and the Sequence generators). These options are used globally, that is you cannot specify these parameters for each class (as the Castor mapping allows). Leave the controls empty to accept default values.

Options at table level can be accessed after selecting a table node and are shown in figure 2.2. These options map directly the Castor elements. Refer to Castor documentation for more information.

Options at field level are shown in figure 2.3. Labels marked as (sql) refer to the sql xml element of the Castor mapping. Note that the use of default values results in a smaller mapping file.

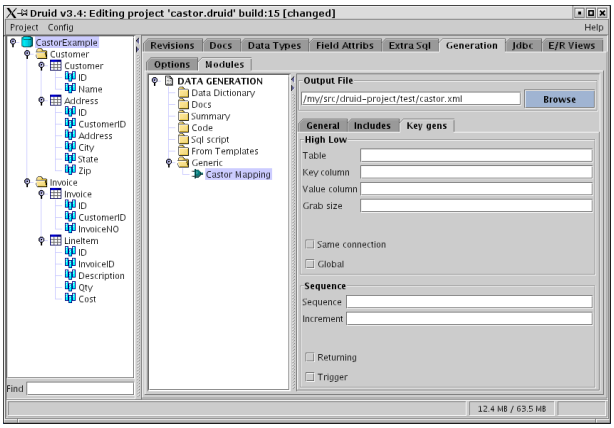

Figure 2.1: The castor module, with options at database level.

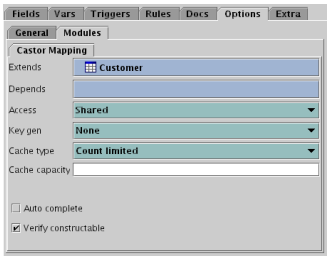

Figure 2.2: Castor options at table level.

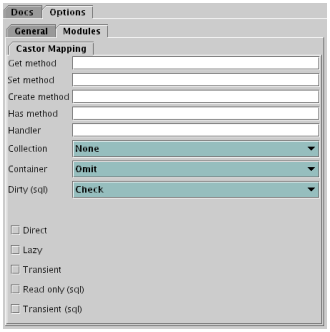

Figure 2.3: Castor options at field level.

2.3 Supported features

2.3.1 Mapping element

Sub element	Status	Auto	Notes
description	ok	yes	Will contain the database documentation.
include	ok	-	
class	ok	semi	
key-generator	ok	-	Supported parameters only for the High/Low and the sequence key generators.

2.3.2 Class element

Sub element	Status	Auto	Notes
description	ok	yes	
cache-type	ok	-	
map-to	ok	yes	Maps to table's name.
field	ok		

Attribute	Status	Auto	Notes
name	ok	yes	Mapped to table's name using package and class suffix.
extends	ok	-	
depends	ok	-	
identity	ok	yes	Mapped to primary keys.
access	ok	-	
key-generator	ok	-	
auto-complete	ok	-	
verify-constructable	ok	-	

2.3.3 Field element

Sub element	Status	Auto	Notes
description	ok	yes	Supplied description for the field.
sql	ok	-	
bind-xml	-	-	
ldap	-	-	

Attribute	Status	Auto	Notes
name	ok	yes	Field's name.
type	ok	yes	Mapped field's basic type (resolved if foreign key).
required	ok	yes	Mapped to not null attrib.
direct	ok	-	
lazy	ok	-	
handler	ok	-	
get-method	ok	-	
set-method	ok	-	
has-method	ok	-	
create-method	ok	-	
transient	ok	-	
container	ok	-	
collection	ok	-	

The type attribute is mapped into a proper java type using the following table:

Druid's type	Mapped to
int4	integer
int8	long
text	string
varchar	string
varchar2	string
nvarchar2	string
number	integer

2.3.4 Sql element

Attribute	Status	Auto	Notes
name	ok	yes	Field's name
type	ok	yes	Mapped field's basic type
many-key	ok	yes	Foreign field (if any).
many-table	ok	yes	Foreign table (if any).
read-only	ok	-	
dirty	ok	-	
transient	ok	-	

The type attribute is mapped into a proper sql type using the following table:

Druid's type	Mapped to
int4	int
int8	long
text	varchar

Chapter 3

The Torque Module

3.1 Introduction

Torque is an Apache project (HTTP://DB.APACHE.ORG/TORQUE/GENERATOR) that takes an XML file describing a database schema and generates the proper SQL instructions to create it. This module creates this XML file exporting the schema from Druid. There is also an import module to import an already existent torque's schema.

3.2 The import module

Do a right click on the project's tree view to show the popup menu and select the Import... item. When the file dialog appears, select the Torque XML Schema file type.

3.3 The generation module

The module is located under the node Generic and is named Torque Schema. The module has options at database, table and field levels. Click the corresponding database, table or field node to edit the options. For table and fields, options are located in the options ▷ modules tab. In figure 3.1 you can see the options available at database level.

3.4 Supported features

Almost all Torque's features are supported. Optional attributes are generated only if their value is not the default one. String attributes are not generated if they are empty. The status column indicates if the attribute is supported. The auto column indicates if the attribute is automatically filled. If not, it can be manually filled using the GUI. Here is a list of supported elements:

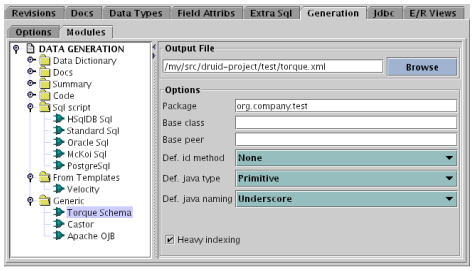

Figure 3.1: The Torque module with options at database level

3.4.1 Database element

Attribute	Status	Auto	Notes
name	ok	yes	
defaultIdMethod	ok	-	
defaultJavaType	ok	-	
package	ok	-	
baseClass	ok	-	
basePeer	ok	-	
defaultJavaNamingMethod	ok	-	
heavyIndexing	ok	-	

3.4.2 Table element

Attribute	Status	Auto	Notes
name	ok	yes	
javaName	ok	-	
idMethod	ok	-	
skipSql	ok	-	
abstract	ok	-	
baseClass	ok	-	
basePeer	ok	-	
alias	ok	-	
interface	ok	-	
javaNamingMethod	ok	-	The default is not specified. Assumed: nochange
heavyIndexing	ok	-	The default is not specified. Assumed: false
description	ok	yes	Line feeds are converted into a dot followed by a space

3.4.3 Column element

Attribute	Status	Auto	Notes
name	ok	yes	
javaName	ok	-	
primaryKey	ok	yes	
required	ok	yes	
type	ok	yes	Druid's basic type
javaType	ok	-	The default is not specified. Assumed: primitive
size	ok	yes	
default	ok	yes	
autoIncrement	ok	-	
inheritance	-	-	
inputValidator	ok	-	
javaNamingMethod	ok	-	The default is not specified. Assumed: nochange
description	ok	yes	Line feeds are converted into a dot followed by a space

3.4.4 Foreign keys

Only simple foreign keys are supported. The name attribute is generated using the template for foreign keys. For the On update and on delete clauses only the following values are supported: none, cascade, setnull. The restrict attribute is not handled by Druid and the set default attribute is not handled by Torque.

3.4.5 Indexes

Indexes are fully supported. Unique indexes are treated as unique elements. The name attribute is generated only for unique indexes using the template named other in the Templates for table constraint names panel.

3.4.6 Unique element

Druid's unique attributes are fully supported both for unique at field level and unique at table level. The name attribute is generated if a template for the unique is present (named other in the GUI panel).

3.4.7 Elements not supported

The following elements are not supported:

- external-schema (database's child)
- id-method-parameter (table's child)

- inheritance (column's child)