Template Job Service Examples

The template job service allows you to create personalized email messages using an email template, and some datasource. In most cases the datasource is a relational database.

Example Database Structure

Lets Start by creating a simple database, this example uses MySQL as the database server, but the code should pretty much work with any database. We are creating 3 database tables:

- `newsletters` — holds the content for each newsletter
- `newsletter_emails` — holds all the email addresses, and personalization (first / last name etc) subscribed to the newsletter
- `templates` — stores templates used for the newsletter.

Here is some DDL/SQL code to create the database tables in mySQL:

```
CREATE TABLE newsletters (
  id int(10) unsigned NOT NULL auto_increment,
  subject varchar(100),
  html_content text,
  template_id int(10) unsigned,
  send_date timestamp,
  text_content text,
  sent int(10) unsigned default '0',
  PRIMARY KEY (id)
);

CREATE TABLE newsletter_emails (
  id int(10) unsigned NOT NULL auto_increment,
  email varchar(200),
  first_name varchar(45),
  last_name varchar(45),
  PRIMARY KEY (id)
);

CREATE TABLE templates (
  id int(10) unsigned NOT NULL auto_increment,
  template text,
  name varchar(45),
  PRIMARY KEY  (id)
);
```

Example Template

You can use any variable you choose to form your database in your template. The variables or tokens are marked as `${variableName}` from within your template.

Note

The format of the template must comply to RFC 821 and RFC 1521

Lets take a look at the example template below:

```
From: Widget News <newsletter@widgets.com>
To: ${first_name} ${last_name} <${email}>
Subject: ${subject}
```

```
MIME-Version: 1.0

Hi ${first_name}

${text_content}

--
Thanks,
Widget Corp
http://widgets.com/

Want us to stop sending you newsletters? Visit the link below:
http://widgets.com/rm.cfm?email_id=${email_id}
```

Note

Note that the template engine will automatically ensure that lines end with \r\n (CRLF) to comply with RFC 821

Template Job Configuration

Once you have your database and your template all set, it's time to configure the XMS Template Job Service in `config.xml` to poll the database for jobs.

The template job service uses the notion of providers to populate variables. The most common provider is the Database Provider, which requires a JDBC driver to connect to a JDBC datasource. Most modern databases provide JDBC drivers. Other providers include the simple provider, which lets you set variables from the `config.xml` file, and a file system provider which can provide variables from files. The provider interface is extensible, so we (or you) can create providers for any type of data source.

Creating a JDBC DataSource in config.xml

The following configuration in `config.xml` will create a datasource mapping to a MySQL database on `127.0.0.1` named `exampledb` with username `user` and password `pass`.

```
<service class="xms.dataservices.XMSDataSourceService">
        <datasource name="example" jdbcdriver="com.mysql.jdbc.Driver"
                jdbcurl="jdbc:mysql://127.0.0.1/exampledb"
                username="user"
                password="pass" />
</service>
```

Note

Before you start the server you need to add the `mysql-driver-file.jar` to your classpath. The best way to do this is by putting it in your `plugins` directory.

Creating a Template Service Job

Next we need to configure the server to check the `newsletters` table for unsent newsletters.

```
<service class="xms.template.XMSTemplateJobService" name="tempalteservice">
  <job interval="1800000" target="spool">
        <provider class="xms.template.provider.DBProvider" datasource="example">
                <sql><![CDATA[
                  SELECT template AS TEMPLATE,
                        n.id AS newsletter_id, subject, send_date, text_content
                  FROM newsletters AS n, templates AS t
                  WHERE sent = 0
                  AND send_date <= NOW()
                  AND n.template_id = t.id
```

```
                  ORDER BY send_date ASC
                  LIMIT 1
                ]]></sql>
                <commitpoint>
                        <action class="xms.template.action.DBAction"
                          event="success" datasource="example">
                                <sql>UPDATE newsletters SET sent=1 WHERE id=?;</sql>
                                <param>${newsletter_id}</param>
                        </action>
                </commitpoint>
        </provider>
        <provider class="xms.template.provider.DBProvider" datasource="example">
                <sql>SELECT id AS email_id, email, first_name, last_name FROM newsletter_emails;</sql>
        </provider>
  </job>
</section>
```

The template used by the Template service must be in a variable called `TEMPLATE` (must be all caps, case sensitive) that's why in the SELECT statement I use the `AS` keyword to create an alias.

Note
Template Variable names are case sensitive, and so are XML tags and attributes.

We use the `commitpoint` to invoke an action. In this case we invoke a `DBAction` which updates the `sent` column in the `newsletters` table to `1`. The commitpoint is run at the end of the dataset. You can also run a commitpoint inside the provider that selects the `newsletter_emails` you could use this mark or log each email has been sent. Just like providers, actions are not limited to database actions, you could have any sort of operation performed here by creating your own custom action.

Note
Note the use of a CDATA section to escape the < in the SQL statement. This is necessary because the config.xml file must be valid XML.

Template Engines

Just about every feature in XMS is pluggable, and the template parsing engine is no exception. You can choose between two stock template parsing engines, or to build your own. The template parsing engine is added to `config.xml` by inserting a `engine` tag under the `job` tag.

Default Token Template Parsing Engine

The default token template engine is invoked by default, you do not need to supply an `engine` tag, unless you want to change the default start and end tokens. The default start token is `${` and the default end token is `}`, which will pickup a token named `variable` like this `${variable}`. If you want to pick up a token like `#variable#` use add the following `engine` tag:

```
<job ...>
  <engine class="xms.template.engine.TokenEngine" tokenstart="#" tokenend="#" />
  <provider ... />
</job>
```

FreeMarker Token Template Parsing Engine

The FreeMarker token template parsing engine allows you to do much more than simply populate variables. With FreeMarker you can create conditional content, loop, format variables, and more - think of it as a mini scripting language for templates.

To enable the FreeMarker Template Engine add the following `engine` tag:

```
<job ...>
  <engine class="xms.template.engine.FreemarkerEngine" />
  <provider ... />
</job>
```

With the FreeMarker template engine enabled you can do things like this:

```
<#if first_name = "Pete">
 We are offering a discount to anyone with the first name of ${first_name}.
 Your Discount code is: 23AFHYST
</#if>
```

The next example uses the modulus operator to send a discount to 1 in 10 people:

```
<#if (email_id % 10) = 0>
 You have been selected to get a discount.
 Your Discount code is: 23AFHYST
</#if>
```

Note

See the FreeMarker Documentation for more information about FreeMarker template syntax.