# Fragments

The HotRod configuration file can be divided into many parts known as fragments. In short, these fragments help to modularize the configuration file and to reduce SCM repository conflicts when multiple developers try to commit changes at the same time.

A simple small project usually has few tables and views, and a single or few developers. In this case the HotRod Configuration file will have a few lines and won't require much effort to modify it. It also won't create many repository conflicts either since there's not much to modify and there aren't many developers in the first place.

However, when the project grows the HotRod configuration file may start to grow. It will grow moderately in terms of tables and views; it will mostly grow, however, when adding arbitrary SQL with the <select> and <update> tags. On one side these are bulky per se. But more importantly, their numbers can become very abundant.

Additionally, the number of developers may also grow, and this can cause a large number of potential SCM repository conflicts while committing changes simultaneously.

Also, when the database can naturally be divided into separate modules, the team may prefer to have the persistence separated by these modules. These way when the persistence changes for a module just that specific part of the configuration is modified and not the whole configuration file.

Fragment files are included in the main configuration file—or in other fragments—by using the <fragment> tag. The <fragment> tag can be intermixed with the <table>, <view>, <select>, and <dao> tags as needed.

Fragment files do not include the header section the main configuration file has. Also note the boilerplate initial lines are different from the main configuration file as well. Fragment files' main tag is hotrod-fragment and not hotrod. Their grammar definition is different accordingly as shown in the DOCTYPE definition.

## Inner Fragments

Fragments can also include fragments of their own. However, the whole tree/subtree of included files through fragments cannot include the same file more than once. This rule also prevents the existence of circular references. If this situation arises HotRod finds out, stops it execution, and shows an error message explaining the details.

## Partial Code Generation

Fragments are not a way of producing partial code generation. They only allow a big configuration file to be separated (and stored in the repository) in multiple smaller files.

In order to perform a HotRod partial generation the team must use *facets*. Facets identify subsections of the configuration file across all fragments. The HotRod Generator can be instructed to generate only one or a subset of facets, reducing the generation time significantly for large projects.

For more details on facets see the Configuration Reference section for Facets.

## Fragment Example

Now, the following configuration file is divided in four files. The main configuration file is shown below:

<?xml version="1.0"?>

<!DOCTYPE hotrod SYSTEM "hotrod.dtd">

<hotrod>

<generators>

<mybatis>

...

</mybatis>

</generators>

<table name="employee" />

<fragment file="web-site/sales.xml" />

<view name="department" />

<fragment file="integrations/suppliers.xml" />

<select java-class-name="ExecutiveEmployee">

<![CDATA[

select \* from employee where is\_executive = 1

]]>

</select>

</hotrod>

The sales.xml fragment, located in the web-site subdirectory relative to the main configuration file, has the following content:

<?xml version="1.0"?>

<!DOCTYPE hotrod-fragment SYSTEM "hotrod-fragment.dtd">

<hotrod-fragment>

<table name="order" />

<table name="gift\_card" />

<fragment file="categories-and-catalogs.xml" />

<select java-class-name="ExpensiveOrder">

<![CDATA[

select \* from order where price &gt; 1000.00

]]>

</select>

</hotrod>

The above file includes a couple of <table> definitions, a <select>, and also includes another fragment. All these tags can be intermixed as needed.

The categories-and-catalogs.xml fragment referenced by the sales.xml fragment is located in the sales directory (not in the main folder), since its location is relative to the sales.xml fragment and it does not specify any relative directory. Its contents is shown below:

<?xml version="1.0"?>

<!DOCTYPE hotrod-fragment SYSTEM "hotrod-fragment.dtd">

<hotrod-fragment>

<table name="catalog" />

<table name="subcatalog" />

<table name="category" />

<table name="subject" />

</hotrod>

The file above includes four tables probably relates to categories management.

Finally, the suppliers.xml fragment is located in the integrations subdirectory relative to the main configuration file. Its contents is shown below:

<?xml version="1.0"?>

<!DOCTYPE hotrod-fragment SYSTEM "hotrod-fragment.dtd">

<hotrod-fragment>

<table name="supplier" />

<table name="inventory" />

<table name="invoices" />

<select java-class-name="OutstandingInvoices30Days">

<![CDATA[

select \* from invoices

where days\_late between 30 and 60

]]>

</select>

</hotrod>

As you see all the fragments can include <table>, <view>, <select>, <dao>, and <fragment> tags. You can intermix them as needed.

The goal of the fragments is to make the whole configuration more manageable, specially when the project grows in number of database object (tables, views, sequences, etc), in terms of optimization, in terms of customized SQL code, number of developers, etc., or a combination of these.