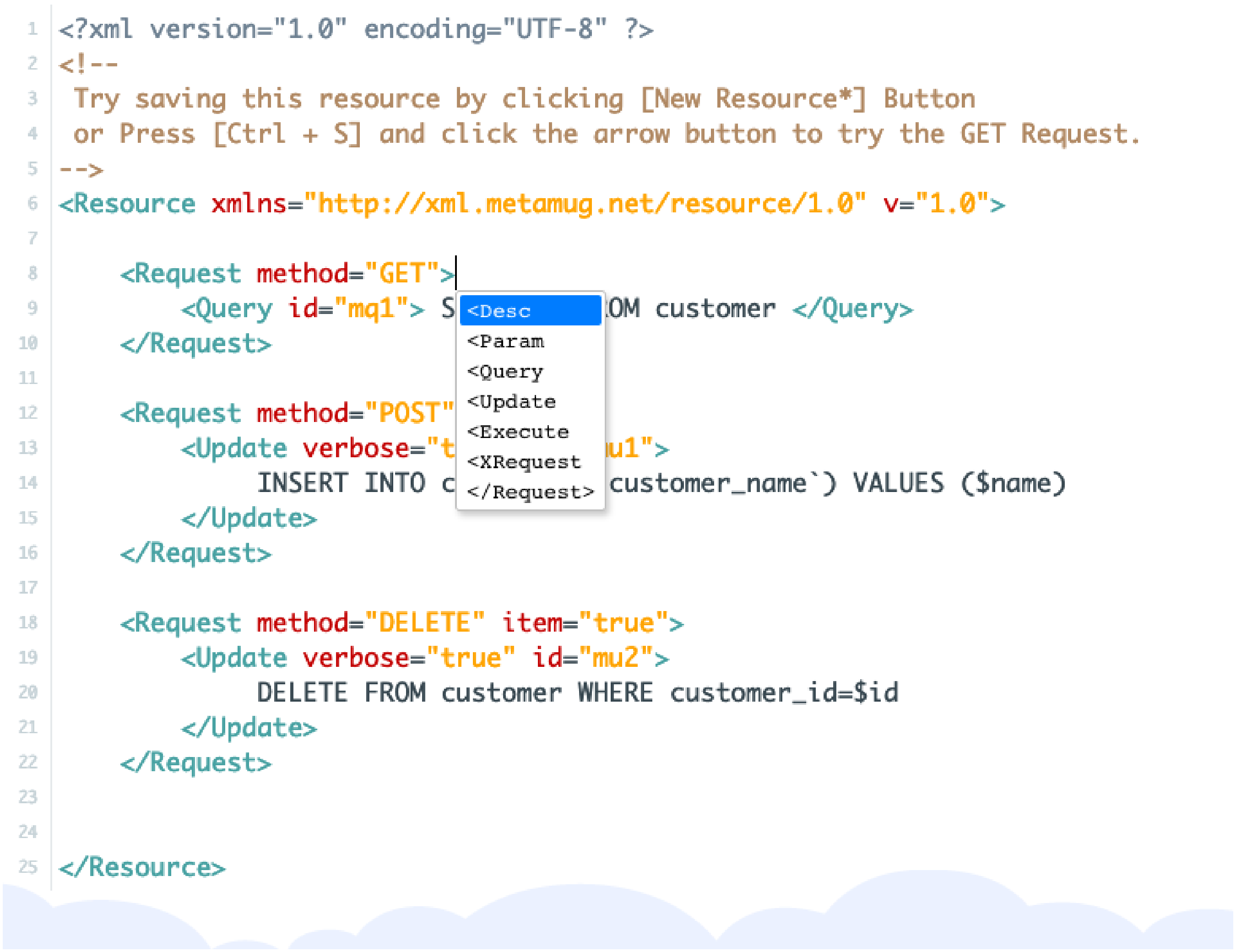
Resource File

Before proceeding to the following article, the reader must be familiar with the following

- XML (Extended Markup Language).
- SQL (Structured Query Language) in relational database management systems.

The REST architectural style considers each data object as a *resource*. The name of the resource name is included in the URL and HTTP requests like GET, POST, PUT, DELETE can be made on the URL endpoint.

Metamug uses XML documents (resource files) to describe the behavior of REST APIs. The files are used to describe the server-side operation to be performed when an HTTP request is made to a particular resource.



```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
 Try saving this resource by clicking [New Resource*] Button
 or Press [Ctrl + S] and click the arrow button to try the GET Request.
-->
<Resource xmlns="http://xml.metamug.net/resource/1.0" v="1.0">

    <Request method="GET">
        <Query id="mq1"> S          OM customer </Query>
    </Request>

    <Request method="POST"
        <Update verbose="t          u1">
            INSERT INTO c          customer_name`) VALUES ($name)
        </Update>
    </Request>

    <Request method="DELETE" item="true">
        <Update verbose="true" id="mu2">
            DELETE FROM customer WHERE customer_id=$id
        </Update>
    </Request>


</Resource>
```

Item Request

Attribute `item="true"` is to indicate that the request is going to be an Item request. Item Id in the URL is directly mapped to **$id** variable in the SQL query.

Here's an example.

Type	URI	$id
Collection	/movie/	null
Item	/movie/22	22

When the request uri suggests an Item Request, request type with `item=true` attribute is executed. And `$id` attribute will be populated with 22. Ofcourse, HTTP method should matched.

resource-file

```
<?xml version="1.0" encoding="UTF-8" ?>
<Resource xmlns="http://xml.metamug.net/resource/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://xml.metamug.net/resource/1.0 http://xml.metamug.net/schema/resource.xsd"
     v="1.0">
    <Desc>Contains information about movies.</Desc>

    <!-- Collection GET Request /movie -->
    <Request method="GET">
        <Desc>Get info of all movies with ratings greater than 3.</Desc>
        <Sql id="wellRatedMovies">
            SELECT name,rating FROM movie
            where rating gt 3.0
        </Sql>
    </Request>

    <!-- Item GET Request /movie/{id} -->
    <Request method="GET" item="true">
        <Desc>Get info of movie with given id.</Desc>
        <Sql id="allMovies">
            SELECT * FROM movie where id=$id
        </Sql>
    </Request>

</Resource>
```

The xml document is used to describe a resource. The title of the resource file ("movie" in this case) is mapped as the name of the particular resource.

> *Avoid the use of $id in a collection request, as it will cause failure.*

Resource Tag

`<Resource>` is the root element of the XML document.

Attributes

- `version` is used to describe the version of the particular resource file, "1.0" in this case. When making an HTTP request to the above resource, the version must be included in the URL in the manner *{baseUrl}/v1.0/movie* that is, prefixed with "v" character. Versioning can be used to implement multiple resource files.
    1. A GET request made to `{baseUrl}/v1.0/movie` will return all the records in the table "movie".
    2. A GET request made to `{baseUrl}/v1.0/movie/1` (for example) will return only the record with id=1 (as id=$id in the SQL query) in the table "movie".
- `auth` the attribute is used for setting an authentication role for the resource. This is explained further in the auth section.

Desc Tag

`<Desc>` is an optional element used for describing the particular resource. The description is shown in the API documentation. `Desc` tag can also be used inside the `Request` tag to describe the request being handled.

Request Tag

1.The first `Request` tag is used to describe an HTTP request made on a collection resource. The children of the resource tag describe the action to be performed after the request is made.

This tag represents a GET request made to `{baseUrl}/movie,` where "movie" is the resource name.

2.The second `Request` tag in the document has a new attribute item which is set to "true". This denotes a request to fetch a particular item represented by the number in the URL.

The URL in this case will look like `{baseUrl}/movie/{id}.` The first Request is a collection request and has no item attribute (item="false" by default) wherein the URL was simply `{baseUrl}/movie.`

Attributes

- `method` describes the HTTP request type - GET/POST/PUT/DELETE. If no method attribute is stated, the method is assumed to be GET by default.
- `status` is an optional attribute used for specifying HTTP status code that should be returned in the response.
- `item` is an optional attribute used for specifying item requests.

Header Tag

Custom HTTP response headers can be set using Header tags as follows

header-tag

```
<Request method="GET">
    <Header name="Powered-By" value="Metamug" />
    <Header name="Cache-Control" value="no-cache" />
</Request>
```

Text Tag

The text tag allows the server to respond with direct information. The simplest use case can be to print a string in the response.

text-tag

```
<Resource xmlns="http://xml.metamug.net/resource/1.0" v="1.0">
    <Desc> Text Tag Example </Desc>
    <Request method="GET">
      <Text id="textoutput"> Hello World </Text>
    </Request>
</Resource>
```

`Sql Tag`

Sql tag holds the relation database sql statement to be executed. The sql inside the tag needs to be passed in plain text or it can be wrapped in CDATA Section. Query can be a join across multiple table or any DML statement.

> *There is no mapping between the resource name ("movie" in this case) and the name of the database table ("movie" in this case). A SQL statement written inside a Request tag is independent of the type of HTTP request or the resource.*

Attributes

- `type` attribute is used to determine the type of the query. It is an indication to the compiler to consider the wrapped sql statement as the assigned type. The `query` type inside the `Sql` tag is used for making SELECT queries. DML commands like **INSERT**, **UPDATE**, **DELETE** etc. must be use type `update` on `Sql` tag.
- `when` attribute is non-mandatory and can have a logical expression as the value. When Request tag contains more than one Query/Update tag, it MUST use when attribute for conditional queries.
- `ref` attribute can be used to refer to SQL query from the SQL Catalog

As a thumb rule, you can say, queries that return a result-set should be enclosed in `<Sql type="query">` tag, rest all in `<Sql type="update">` tag the later is mostly DML queries (INSERT, UPDATE, DELETE), but there is one caveat here for PostgreSQL there is a keyword called `Returning` which can return a result even though we're using DML queries. Now there is no `RETURNING` keyword in MySQL or MS-SQL (yet), but if you happen to simulate such a feature on your own, do enclose it under `<Sql type='query'>` tag.

XML Escape Characters

In the movie.xml example above, the first SQL was written is `SELECT name, rating FROM movie where rating gt 3.0`. The characters `gt` in this SQL represent the > (greater than) symbol. Thus, the actual SQL executed will be `SELECT name, rating FROM movie where rating > 3.0`. The XML format does not support the following symbols and they are required to be escaped using characters as shown below:

Symbol	Escape Characters
>	gt
>=	ge
<	lt
<=	le
=	eq
!=	ne

Updating Data

Insert, Update and Delete operations can be performed on the database using POST, PUT and DELETE requests.

> *This is only a convention. It is not enforced by design. Any type/number of SQL statements can be added under a single request tag.*

movie.xml

movie.xml

```
<Request method="POST" status="201">
    <Sql id="addMovie">
        INSERT INTO movie (name, rating) VALUES ($p, $q)
    </Sql>
</Request>

<Request method="PUT" status="202" item="true">
    <Sql id="updateMovie">
        UPDATE movie SET rating=$rating WHERE id=$id
    </Sql>
</Request>

<Request method="DELETE" status="410" item="true">
    <Sql id="deleteMovie">
        DELETE FROM movie WHERE id=$id
    </Sql>
</Request>
```

`POST, PUT, DELETE requests for the resource can be added as shown above.`

1. POST request is made on a resource collection. The SQL statement contains two variables $p and $q. Here, "name" and "rating" are the column names of the table. A POST request made to `{baseUrl}/movie` with parameters `p=Madagascar&q=4.5` will add a new movie record with the name as Madagascar and rating 4.5. The parameter names p and q are directly mapped with variable names $p and $q.
2. PUT request is made on a resource item. In above example, a PUT request made to `{baseUrl}/movie/2` with request parameter `rating=4` will update the entry in the "rating" column, of the record with `id=2`, with the value 4.
3. DELETE request is made on a resource item. A DELETE request made to `{baseUrl}/movie/3` (for example) will delete the record with `id=3.`

Conditional Queries with when

Multiple elements can be used inside a `Request` tag. In such cases when attribute should be used in the Sql elements. `when` attribute evaluates a logical expression. On the successful evaluation of the expression, the SQL statement is executed.

Multiple SQL statements can be executed on the same request. If their respective "when" expressions evaluate to true. e.g `when="$amount lt 10000", when="$type eq 'one' and $color eq 'blue'"`

movie.xml

conditional-queries

```
<?xml version="1.0" encoding="UTF-8" ?>
<Resource v="1.2" xmlns="http://xml.metamug.net/resource/1.0">
    <Desc>Contains information about movies.</Desc>

    <Request method="GET">
        <!-- SQL 1 -->
        <Sql id="list" when="$q eq 'all'">
            SELECT name,rating FROM movie
        </Sql>

        <!-- SQL 2 -->
        <Sql id="mostRated" when="$q eq 'maxRated'">
            SELECT name,rating FROM movie
            where rating=5.0
        </Sql>

        <!-- SQL 3 -->
        <Sql id="lowRated" when="$q eq 'midRated'">
            SELECT name,rating FROM movie
            where rating lt 3.0
        </Sql>
    </Request>
</Resource>
```

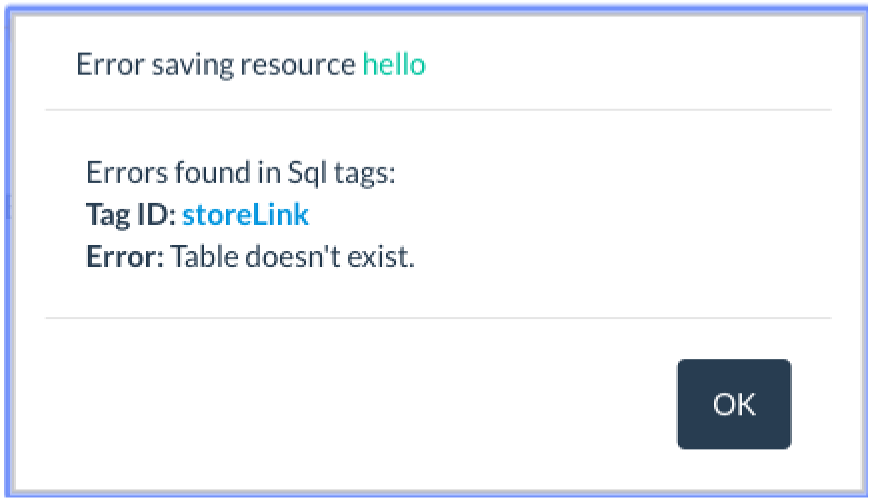
`In the above example, there are three Sql elements used inside a Request. A query parameter q sent in the HTTP request will determine which of the three SQL queries will be executed and the corresponding result will be returned. SQL 1 will be executed when a request is made with URI {baseUrl}/v1.2/movie?q=1 as determined by when="$q eq 1" and so on.`

Its more readable to use strings wherever possible to represent your sql like `when="$q eq 'all'"` or `when="$q eq 'sort_name'"`.

Mpath expressions also use when condition for evaluating and selecting request elements to be executed.

SQL Query Testing on Resource Save

On saving the resource, the SQL queries written inside the resource are tested internally. If errors are found, the resource is not saved and the errors are shown in the console.



Error saving resource hello

Errors found in Sql tags:
Tag ID: storeLink
Error: Table doesn't exist.

On successful testing, the SQL queries are saved by the system and can be viewed in the SQL Catalog.

Calling Stored Procedures

Consider a stored procedure defined as shown

calling stored procedures

```
CREATE PROCEDURE HelloName(IN name VARCHAR(100))
BEGIN
    SELECT CONCAT("Hello ", name);
END
```

`This stored procedure can be called inside the resource file as follows`

```
<Request method="GET">
     <Sql id="myQuery" type="query" > CALL HelloName($name) </Sql>
</Request>
```

Here, the value of $name can be passed as *name=John* in the request.