2.0 DelayFeeder API Middleware

2.1 Theory of Operation

The DelayFeeder application and API is a special piece of middleware that keeps a list of service alert RSS feeds updated, exposing the results via JSONP.

When a request comes in for a list of routes, DelayFeeder checks to see how old its cached copy of the backing RSS feed is, and if it's older than 15 minutes, requests the feed from the origin server. When the response comes in, or if the cached copy is less than 15 minutes old, the system returns the latest item in the backing RSS feed as a JSONP data structure.

DelayFeeder has a backing database to store state, but can be successfully balanced with a load balancer. No user session information/passing is needed.

2.2 Configuration

2.2.1 Database Connection Information

DelayFeeder has a backing database to store the RSS service alert feeds and the latest copy of each feed. Depending on the environment, you may need to change the database username/password, hostname or database name. The process below identifies how to do that. The instructions below assume you are running the Tomcat servlet container.

1. Find the running DelayFeeder webapp in your Tomcat “webapps” directory (this is usually named “delayfeeder”).

If this directory does not exist, uncompress the “delayfeeder.war” file with an unzipping tool into the webapps directory—the directory created should be called “delayfeeder” (with two subdirectories called “WEB-INF” and “META-INF”).

> NOTE: If you modify the deployed delayfeeder webapp (as opposed to the .war file directly), your changes may be inadvertently lost during a redeployment of the webapp. For this reason, we recommend that you remove the “delayfeeder.war” file from the Tomcat webapps directory to ensure any running delayfeeder webapp honors the changes you will be making as part of this process!

2. Within delayfeeder directory, find the WEB-INF directory.

3. Within the WEB-INF directory, find the classes directory.

4. Within the classes directory, find the data-sources.xml file.

5. Inside the data-sources.xml file, there are three parameters that may need to be changed (in bold below):

```
<property name="url" value="jdbc:postgresql://localhost/org_soundtransit_delayfeeder" />
<property name="username" value="postgres" />
<property name="password" value="changeme" />
```

They are, in order:

- Database connection string. This is of the format jdbc:postgresql://<database host>/<database name>. The database should be a running PostgreSQL database that already exists.
- Database username: the username with which to connect to PostgreSQL.
- Database password: the password for the username above.

2.2.2 Updating the Backing Service Alert RSS Feeds

Internally, DelayFeeder maintains a list of RSS feeds and routes that they contain service alerts for. To update this list of known feeds you'll need a copy of the source code available at https://github.com/chrispatterson/soundtransit-planner, as well as a system with Apache Maven 2 installed.

To update the list of backing RSS feeds:

1. Find the delayfeeder directory that you checked out from the above Git repository. This directory should have "README" and "feeds.txt" files within it.

2. Ensure the "feeds.txt" file contains all of the feeds you want DelayFeeder to know about. The format of the file is operator, designator and RSS feed URL, separated by commas, one per line.

Example feeds.txt contents:

```
MT,542,http://service.govdelivery.com/service/rss/item_updates.rss?code=WAKCDOT_279
MT,545,http://service.govdelivery.com/service/rss/item_updates.rss?code=WAKCDOT_259
MT,550,http://service.govdelivery.com/service/rss/item_updates.rss?code=WAKCDOT_207
MT,554,http://service.govdelivery.com/service/rss/item_updates.rss?code=WAKCDOT_208
[... snip ...]
```

3. Once you have updated the feeds.txt file with all feeds DelayFeeder should know about, run:

```
mvn exec:java -Dexec.mainClass=org.openplans.delayfeeder.LoadFeeds -Dexec.args=/<path
to feeds.txt created in step 2>
```

Maven will check for updates to itself, and run the update process. If the database configured in data-sources.xml is empty, it will create the schema and populate the feed list.

4. You can check the feed import process was successful by connecting to the PostgreSQL database you configured in data-sources.xml and running:

```
select * from route_feed
```

You should see the feeds you entered in feeds.txt in the query result.

2.3 API Documentation

2.3.1 status Method

This method accepts a list of operators and designators, and returns a JSON(P) data structure containing the aggregate service alerts.

Input parameters:

- route: Comma separated pair of operator and designator. This element can appear as many times as necessary to specify each route alerts should be returned for. Example: ST,524.
- callback (optional): Sets the JSONP callback function included in the response.

Example request REST URL:

```
http://sea.dev.openplans.org:8080/delayfeeder/ws/status?callback=getAlerts127151228&route=ST,594&route=ST,574&_1303320559691=
```

Example response:

```
getAlerts127151228({"items":{"item":[{"agency":"ST","route":"594","link":"http://stageredesign.soundtransit.org/Schedules/Alerts/Route-594-reroute-315.xml","status":"Reroute due to St Patricks day parade","category":"Yellow","date":"2011-04-20T10:28:50-07:00"},{"agency":"ST","route":"574"}]}})
```

Each “item” (service alert) consists of an agency identifier (“agency”), a route designator (“route”), a URL to the service alert (“link”), the service alert title (“service”) and lastly the category (“category”). The date of the last refresh of the backing RSS feed is also returned in the “date” field. An “item” is returned for each agency and route requested, even if there are no service alerts available (see example above).