RSS API

Francisco Berrizbeitia

CONCORDIA UNIVERSITY LIBRARY

Contents

Overview

This API was developed to replace the Google feed API that's been officially deprecated. The main purpose of this development is to provide the means to consume and process RSS feeds using javascript only. This cannot be made using straight javascript because of cross domain restrictions. To overcome this the API uses cross-origin resource sharing (CORS), allowing access to calls from a domain white list.

Besides implementing the CORS mechanism the system also caches the feeds to improve reliability and performance. The locally stored files are preprocessed to clean up the non-standard tags and sort item on the given criteria. An index of all cached is maintained. Once the feed is included in the index a timed script (cron job) refreshes the content regularly.

Log files for both feed updates and access are maintain also to allow in depth analysis on the use and performance of the system.

In the following sections we'll provide a more detailed description of the aforementioned functionalities: feed retrieval, cashing and logging.

Functional Description

Feed retrieval

After a call is made using the javascritpt client-side library the sever side script will first check if the call was made from a safe domain, if so, will check if the feed is cached and up to date before returning it. If the request file is not in the index, the script will get file in the origin, cache it and add it to index before returning the locally store data. This process may take a few seconds causing the first call to a feed to be slower. In Algorithm 1 bellow is the pseudo code of the implementation for this process.

```
IF (domain_allowed)
        IF (feed_is_cached)
                IF (feed_is_up_to_date)
                        RETURN cached_feed
                ELSE
                        update_feed
                        RETURN cached_feed
                END IF
        ELSE
                Cache feed
                RETURN cached_feed
        END IF
ELSE
        RETURN FALSE
END IF
```

Algorithm 1. Pseudo code for feed retrieval

Caching

The cache is created as new calls are made. When a new feed is requested an identifier is created using md5 hashing function with the url as input, this is done to avoid filenames problems within the file system.

Before creating the cached file the script tries to "standardize" the RSS feed. This means that it would try generate the cached feed using the core standard tag for the following items: <title>, <description>, <link>, <guid>, <pubdate>, <category>, <enclosure>. Additional fields from extended vocabularies are left intact as well as the channel information and the vocabularies declarations. Sorting also occurs before storing the file.

An index is maintained with the following information for each feed (see Figure 1):

- **url:** The feed's location
- **limit:** number of items to include in the cached file
- **id:** the local filename created by hashing the url.
- **lastUpdated:** Last time the local file was updated
- **sort:** sorting parameters, if left blank to sorting is performed
    - **field:** name of the tag
    - **type:** data type of the sorting field. Can be either: string, numeric or date.
    - **order:** Sorting order, either asc or desc.

```
<feed>
     <url>http://rss.sciencedirect.com/publication/science/00018708%20</url>
     <limit>25</limit>
     <id>a167935c74de0cba4c3153b19ada3a31.xml</id>
     <lastUpdated>Wed, 13 Jul 2016 03:01:07 -0400</lastUpdated>
     <sort>
       <field>pubDate</field>
       <type>date</type>
       <order>desc</order>
     </sort>
</feed>
```

Figure 1.Example: feed information stored in the index

A schedule task (/service/cron.php) is responsible for keeping the feeds in the index up to date. If for some reason the update task is not performed, the feed will be updated in real time.

To clear the cache delete all entries in /service/index.xml and all the files in the /cache/feeds/ directory. Performing this action will not interrupt the service, however it may affect performance for the users temporarily while the cache is being re-created.

Logging

The system creates two sets of log files: requests and updates. The request log contains the timestamp and the id of requested feed separated by a '|', one request per line. A new file is created every day.

The update log is created when the schedule script is called. Each line show the timestamp, id, url and whether the feed was successfully updated or an error occurred during the operation. Is recommended to look at this files to detect broken links.

Files and Directory structure

The files are distributed in three directories:

- Cache: Feed files and the log files.
- Lib: The javascript library.
- Service: the server side processing scripts and the index files.

Installation and Configuration

To install the service copy all files to a directory in your web server, ideally with its own domain. The service and cache folder and subfolders must have writing permission.

In the service folder open config.php, add the allowed domains and the base url for the feeds folder. When adding domains is important to notice that subdomains must be added explicitly to work, the url must include the protocol (http or https) if a domain may access the feeds using either two entries must be included. There's no limit to the number of allowed domains.

In the lib folder open the feed.js file and set the proper values for the serviceURL and the loaderURL. For example:

```
function rss(){
var itemsList = [];
        var serviceURL = 'https://rssapi.example.com/service/RSSget.php';
        var loaderURL = 'https://rssapi.example.com/lib/loading_50.gif';
…
```

Usage

For CORS security reasons, before using the API, the domain and/or subdomains where the /lib/feed.js is included must be whitelisted in the server.

1. Include jQuery.
2. Include the properly configured feed.js file
   EG: `<script src=" https://rssapi.example.com/lib/feed.js "></script>`
3. Create the object: var objRss = new rss();

4. Make a call to the displayAsinc function
5.
```
objRss.displayAsinc(in_url, in_containerID, in_sorting,in_type,
in_sorting_order,in_count,in_descriptionLength);
EG: objRss.displayAsinc(feed,spanId, 'pubDate','date','desc',count, 512);
```

The library provides to ways of dealing with the retrieval and display of the feed: The asynchronous method, as the one show in the example above and set of method for loading and manipulating the feed.

The displayAsinc functions receives the following parameters:

- **in_url:** feed URL
- **in_containerID** : ID of the container where items are going to be created (DOM elements such as DIV)
- **in_sorting:** to sort elements by, for example, title, date (pubdate, dc:date). The function can accept any tag present inside the <item>.
- **in_type:** The variable type of the sorting field: date, numerical or string
- **in_sorting_order:** Either desc or asc.
- **In_count:** The max number of <items> to display.
- **in_descriptionLength:** the max number of characters to display in the description

The function will automatically create all DOM elements to display the feed nested inside the provided container.