

FeedTool

FeedTool is built on the [Rome API](#) and provides the ability to retrieve and manipulate RSS and Atom feeds from within Velocity templates.

The tool works by retrieving a feed from a specified URI and returning it in a [ContextFeedWrapper](#) class. The idea with the wrapper class is to have a convenient location for the provision of convenience methods for feed and feed entry manipulation ... although it could easily be incorporated into a single class.

The tool consists of:

- The [Rome library](#)
- A tool for the toolbox:

```
package org.apache.velocity.tools.view.tools;

import java.net.URL;
import java.net.HttpURLConnection;
import java.io.IOException;
import java.net.MalformedURLException;

import sun.misc.BASE64Encoder;

import com.sun.syndication.feed.synd.SyndFeed;
import com.sun.syndication.io.SyndFeedInput;
import com.sun.syndication.io.FeedException;
import com.sun.syndication.io.XmlReader;
import com.sun.syndication.fetcher.FeedFetcher;
import com.sun.syndication.fetcher.FetcherException;
import com.sun.syndication.fetcher.impl.FeedFetcherCache;
import com.sun.syndication.fetcher.impl.HashMapFeedInfoCache;
import com.sun.syndication.fetcher.impl.HttpURLFeedFetcher;

/**
 * FeedTool
 * <p>
 * A Velocity View Tool for working with RSS and Atom APIs using Rome
 * </p>
 * @author C.Townson
 */
public class ViewFeedTool {
    /**
     * The URL of the feed to be fetched by RomeFetcher
     */
    private URL feedURL;

    /**
     * The FeedObject returned from fetch by RomeFetcher
     */
    private ContextFeedWrapper feed;

    /**
     * Default view tool no-arg constructor
     */
    public ViewFeedTool() {
        // do nothing - View Tools must have public no-arg constructor
    }

    /**
     * Returns the feed located at the specified URL
     * <p>
     * This is the core method provided with this tool - use this
     * from template, rather than any of the other public methods
     * </p>
     * TODO overloaded method with authentication parameters
     * @param feedURL the url of the feed to retrieve
     * @return feed The feed object inside a utility wrapper class
     */
    public ContextFeedWrapper get(String feedURL) {
        // cast String to URL
        this.setFeedURL(feedURL);
    }
}
```

```

        // rev up RomeFetcher
        FeedFetcherCache feedInfoCache = HashMapFeedInfoCache.getInstance();
        FeedFetcher feedFetcher = new HttpURLFeedFetcher(feedInfoCache);

        // retrieve feed or freak out (silently!)
        try {
            // grab feed
            this.setFeed(feedFetcher.retrieveFeed(this.getFeedURL()));
        } catch (FetcherException fetchx) {
            // do something here?
        } catch (FeedException feedx) {
            // do something here?
        } catch (IOException iox) {
            // do something here?
        } catch (Exception x) {
            // do something here?
        }

        // return wrapped feed
        return this.getFeed();
    }

    /**
     * Returns the feed located at the specified (password protected) URL
     * @param feedURL the URL of the feed to retrieve
     * @param userName the userName to access the protected URL with
     * @param password the password to access the protected URL with
     * @return feed the feed object inside a utility wrapper class
     */
    public ContextFeedWrapper get(String feedURL, String userName, String password) {
        // set feed url
        this.setFeedURL(feedURL);

        // construct auth string
        String auth = userName + ":" + password;

        // initialize httpcon variable
        HttpURLConnection httpcon = null;

        try {
            // create connection
            httpcon = (HttpURLConnection) this.getFeedURL().openConnection();

            // encode username:password
            String encoding = new BASE64Encoder().encode(auth.getBytes());

            // set request property
            httpcon.setRequestProperty("Authorization", "Basic " + encoding);

            // connect
            httpcon.connect();
        } catch (IOException iox) {
            // do something here?
        }

        // rev up Rome (not using fetcher this time)
        SyndFeedInput input = new SyndFeedInput();

        // grab and wrap feed
        try {
            this.setFeed(input.build(new XmlReader(httpcon)));
        } catch (FeedException fx) {
            // do something here?
        } catch (IOException iox) {
            // do something here?
        }

        // if we have a connection, disconnect now
        if(httpcon != null) {
            httpcon.disconnect();
        }
    }

```

```

        // return wrapped feed
        return this.getFeed();
    }

    /**
     * @return Returns the feed.
     */
    public ContextFeedWrapper getFeed() {
        return this.feed;
    }

    /**
     * @param feed The feed to set.
     */
    public void setFeed(ContextFeedWrapper feed) {
        this.feed = feed;
    }

    /**
     * @param feed The feed to set
     */
    public void setFeed(SyndFeed feed) {
        this.setFeed(new ContextFeedWrapper(feed));
    }

    /**
     * @return Returns the feedURL.
     */
    public URL getFeedURL() {
        return feedURL;
    }

    /**
     * @param feedURL The feedURL to set.
     */
    public void setFeedURL(URL feedURL) {
        this.feedURL = feedURL;
    }

    /**
     * Private helper method to covert String to URL
     * @param url
     */
    public void setFeedURL(String url) {
        // try to cast URL string to URL
        try {
            this.setFeedURL(new URL(url));
        } catch (MalformedURLException mfux) {
            // do something here?
        } catch (Exception x) {
            // do something here?
        }
    }
}

```

- A wrapper class for feeds

```

package org.apache.velocity.tools.view.tools;

import com.sun.syndication.feed.synd.SyndFeed;
import com.sun.syndication.feed.synd.SyndEntry;

/**
 * Feed wrapper for Velocity context providing
 * utility methods for accessing feed object properties
 *
 * @author Christopher Townson
 */

```

```

*/
public class ContextFeedWrapper {

    /**
     * The feed
     */
    private SyndFeed feed;

    /**
     * An interger containing the total number of items in the retrieved feed
     */
    private int numberOfEntries;

    /**
     * A SyndEntry object for holding the most recent feed entry
     */
    private SyndEntry latest;

    /**
     *
     * @param feed
     */
    public ContextFeedWrapper(SyndFeed feed) {
        // assign feed object
        this.setFeed(feed);

        // grab number of entries
        this.numberOfEntries = this.getFeed().getEntries().size();

        // grab most recent entry (presently: just first entry)
        // TODO make this loop through entries and compare dates
        this.latest = (SyndEntry) this.getFeed().getEntries().get(0);
    }

    /**
     * @return Returns the feed.
     */
    public SyndFeed getFeed() {
        return this.feed;
    }

    /**
     * @param feed The feed to set.
     */
    public void setFeed(SyndFeed feed) {
        this.feed = feed;
    }

    /**
     * @return Returns the numberOfEntries.
     */
    public int getNumberOfEntries() {
        return numberOfEntries;
    }

    /**
     * @param numberOfEntries The numberOfEntries to set.
     */
    public void setNumberOfEntries(int numberOfEntries) {
        this.numberOfEntries = numberOfEntries;
    }

    /**
     * @return Returns the latest feed entry.
     */
    public SyndEntry getLatest() {
        return latest;
    }

    /**
     * @param latest The entry to set as the latest

```

```
        */
        public void setLatest(SyndEntry latest) {
            this.latest = latest;
        }
    }
}
```

- A toolbox.xml entry

```
<!-- FeedTool: for fetching, parsing, and otherwise handling syndicated feeds of any type -->
<tool>
    <key>FeedTool</key>
    <scope>request</scope>
    <class>com.nature.velocity.tools.view.tools.ViewFeedTool</class>
</tool>
```