Using a Custom Heritrix Configuration File

The Heritrix Connector supports configuration files of Heritrix 3.1. Heritrix User Guide for 3.1 describes in detail the different configuration options that can be set in the configuration file.

Contents:

A sample configuration file can be found here: Crawler-beans.xml

The main difference with a standard Heritrix 3.1 configuration file, is that for Aspire the configuration file will need to reference the AspireHeritrixProcessor component instead of the WARCWriterProcessor in the disposition chain.

```
<bean id="aspireProcessor" class="com.searchtechnologies.aspire.components.heritrixconnector.
AspireHeritrixProcessor"/>

<bean id="dispositionProcessors" class="org.archive.modules.DispositionChain">
  <property name="processors">
    <list>
      <ref bean="aspireProcessor"/>
      <ref bean="candidates"/>
      <ref bean="disposition"/>
    </list>
  </property>
</bean>
```

The Heritrix Connector Assume that the Heritrix Engine compute an unique digest for each URL crawled, so the following bean must be configured:

```
<bean class="org.archive.modules.fetcher.FetchHTTP" id="fetchHttp">
  <property name="digestContent" value="true" />
  <property name="digestAlgorithm" value="md5" />
</bean>
```

Basic Authentication

This approach works on both https and http. In this case we are trying to crawl https://myAuthenticatedSite.com/. The configuration would be as follow on the crawler beans:

```
<bean id="credential" class="org.archive.modules.credential.HttpAuthenticationCredential">
  <property name="domain">
    <value>myAuthenticatedSite.com:443</value>
  </property>
  <property name="realm">
    <value>My Authenticated Site Web Browsing</value>
  </property>
  <property name="login">
    <value>mylogin</value>
  </property>
  <property name="password">
    <value>myPassword</value>
  </property>
  <property name="usingNtlm">
    <value>false</value>
  </property>
</bean>

<!-- CREDENTIAL STORE: HTTP authentication or FORM POST credentials -->

<bean id="credentialStore" class="org.archive.modules.credential.CredentialStore">
  <property name="credentials">
    <map>
      <entry key="example-credential" value-ref="credential"/>
    </map>
  </property>
</bean>
```

Remember to customize the fetchHttp as explained at the bottom.

Realm: The realm string must exactly match the realm name presented in the authentication challenge served by the web server. This is obtained forcing a 401 response from the server. Using curl: curl -ik --user mylogin https://myAuthenticatedSite.com/ The console will prompt for a password so you give an incorrect pass and you will obtain something very similar to this:

```
$ curl -ik --user mylogin https://myAuthenticatedSite.com/
Enter host password for user 'mylogin':
HTTP/1.1 401 Authorization Required
Date: Mon, 03 Jun 2013 19:36:05 GMT
Server: Apache/2.2.16 (Debian)
WWW-Authenticate: Basic realm="My Authenticated Site Web Browsing"
Vary: Accept-Encoding
Content-Length: 501
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>401 Authorization Required</title>
</head><body>
<h1>Authorization Required</h1>
<p>This server could not verify that you
are authorized to access the document
requested.  Either you supplied the wrong
credentials (e.g., bad password), or your
browser doesn't understand how to supply
the credentials required.</p>
<hr>
<address>Apache/2.2.16 (Debian) Server at myAuthenticatedSite.com Port
 443</address>
</body></html>
```

The realm should be the one in double quotes on this line **WWW-Authenticate: Basic realm="My Authenticated Site Web Browsing"**

NTLM Authentication

Aspire Heritrix Connector uses a custom Heritrix Engine that was improved in order to handle NTLM authentication.

Example bean configuration

```
<bean id="credential"
  class="org.archive.modules.credential.HttpAuthenticationCredential">

    <property name="domain">
        <value>myNtlmSite.domain</value> <!- if your site uses https your domain will look like this:
myNtlmSite.domain:443 -->
    </property>

    <property name="realm">
        <value>My Authenticated Site Web Browsing</value> <!-- If no realm is used by NTLM leave empty -->
    </property>

    <property name="ntlmDomain">
        <value>myNtlmDomain</value>  <!-- If NTLM requires users to specify a domain, otherwise leave empty
-->
    </property>

    <property name="ntlmRealm">
        <value>My Authenticated Site Web Browsing</value> <!-- If no realm used by NTLM leave empty -->
    </property>

    <property name="login">
        <value>myUsername</value>
    </property>

    <property name="password">
        <value>myPassword</value>
    </property>

    <property name="usingNtlm">
        <value>true</value>
    </property>
 </bean>
```

The realm values can be retrieved from a curl command execution. See previous section for more information.

```
<bean id="credentialStore"
  class="org.archive.modules.credential.CredentialStore">
  <property name="credentials">
    <map>
      <entry key="example-credential" value-ref="credential"/>
    </map>
  </property>
</bean>

<bean class="org.archive.modules.fetcher.FetchHTTP" id="fetchHttp">
  <property name="credentialStore">
    <ref bean="credentialStore"/>
  </property>
  <property name="digestContent" value="true" />
  <property name="digestAlgorithm" value="md5" />
  <property name="sendConnectionClose" value="false" />
</bean>
```

HtmlFormCredential reconnection on expired cookies

Sometimes the servers emit cookies with an expiration time, so if you want to force a reconnection you can do two things:

1. Force a reconnection after a period of time (expireAfter)
2. Force a reconnection for a URL when its content contains a regex (expiredContent)

Example:

```
<bean id="credential" class="org.archive.modules.credential.HtmlFormCredential">
  <property name="expireAfter" value="1500"/>
  <property name="expiredContent" value="window.location = &quot;signup.php&quot;" />
</bean>
```

Enable XSL Transformation

Enable XSLT in Heritrix to allow the engine to extract links and content from the XSLT generated HTML. Also the Aspire Heritrix Connector will use the generated HTML to extract the data to be indexed.

To enable XSL transformations add the following property to the **FetchHTTP** bean

```
<bean class="org.archive.modules.fetcher.FetchHTTP" id="fetchHttp">
    <property name="enableXslt" value="true" />
</bean>
```

By default if the property **enableXslt** is not present, Heritrix will not perform any XSL transformation and will process every xml document as it is.

Crawling Large Web Pages

Heritrix by default sets a maximum of 6000 links to extract from a single URL, the rest of the links found are discarded an therefore not crawled. If your site contains pages with more than 6000 links per URL, you would want to configure the maximum of links to extract. To do so you have to add the following property to the **org.archive.crawler.frontier.BdbFrontier** bean:

```
<property name="maxOutlinks" value="10000"/>
```

so your bean would look like this:

```
<bean class="org.archive.crawler.frontier.BdbFrontier" id="frontier">
  <property name="retryDelaySeconds" value="20"/>
  <property name="maxRetries" value="5"/>
  <property name="maxOutlinks" value="10000"/>
</bean>
```

If you are crawling a web site with large lists of links and a "Next Page" link, you also would like to increase the maximum Hops to do. For example if your web site consists of the following pages of a list of lninks: page1 -> page2 -> ... -> page100, and your max hops is configured as 5, you would get up to the 6th page, so you would want to increase it to at lease 100.

You can configure the max hops like this:

```
<bean class="org.archive.modules.deciderules.TooManyHopsDecideRule">
  <property name="maxHops" value="100"/>
</bean>
```

Configuring Concurrent Connections to the same hostname

By default heritrix uses only one queue per hostname so only one connection would be retrieving data from the same web server. This makes sense on very wide web crawls, but not if you want to crawl a single web site. You can use the Heritrix parallelQueues to have more than one concurrent connection retrieving the data.

There are several ways to distribute the URLs among the parallel queues:

- **SurtAuthorityQueueAssignmentPolicy**

    - This will group the URLs into queues based on its SURT form. The http://www.searchtechnologies.com/ becomes **com, searchtechnologies,www,** so each X.searchtechnologies.com domain will have its own queue
- **BucketQueueAssignmentPolicy**
    - This will group the URLs based on its IP addresses.
- **HostnameQueueAssignmentPolicy**
    - This will group the URLs based on the hostname:port evident in the URL
- **IPQueueAssignmentPolicy**
    - This will group the URLs based on its IP addresses, if there is no IP address available it behaves as the **HostnameQueueAssignmentPolicy**
- **HashingQueueAssignmentPolicy**
    - This will group the URLs evenly based on this formula: urlHash % parallelQueuesSize.

```
<bean class="org.archive.crawler.frontier.BdbFrontier" id="frontier">
  <property name="retryDelaySeconds" value="2"/>
  <property name="maxRetries" value="10"/>
</bean>
<bean id="queueAssignmentPolicy"
   class="org.archive.crawler.frontier.HashingQueueAssignmentPolicy">
  <property name="parallelQueues" value="50" />
</bean>
```

The maxRetries property must be raised if you increase the number of parallelQueues as sometimes it fails at the beginning of the crawl.

Remember to be cautious about how many connections you will use per hostname as that can cause problems to the crawled web site, and it can be considered to be an attack by the web admin.