WCI Feed Service Servlet

Introduction

Objective

Since Web Content Integrator Feed Service Servlet is not currently documented by the formal product wiki, this documents helps developers and designers to understand how and when to use this Servlet.

Document Scope

This document focus only the Feed Service Servlet part of the whole Web Content Integrator solution, that is shipped with WebSphere Portal starting of version 6.5.1 and onwards.

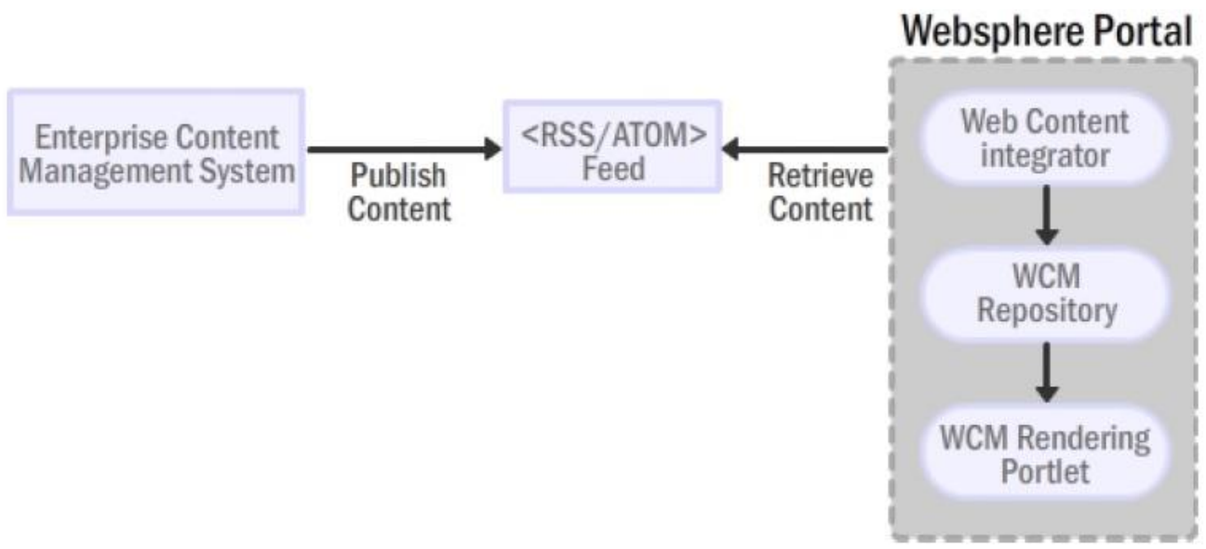
The document assumes that you have the latest Portal version installation with the latest cumulative fixes. At the time writing this document, there was v8.0.0.0 and a latest interim fix of (8.0.0.1-WP-WCM-Combined-CFPM94847-Server-CF08) issued in Oct 8, 2013

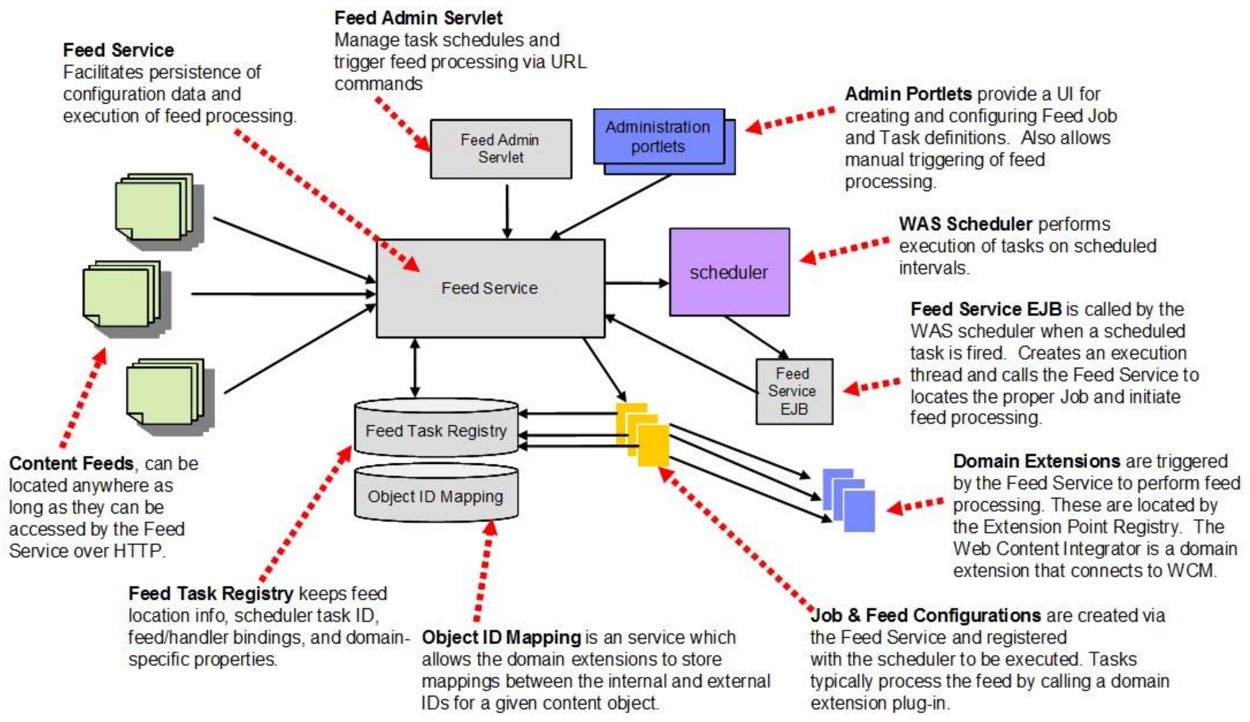
Here are the recommended fixes for WebSphere Portal (8.0.0.0, Linux) from Fix Central, where we should install the latest releases.

Web Content Integrator

The Web Content Integrator is a solution for integrating externally managed Web content with WebSphere Portal. Through the use of standard content syndication feed technologies based on RSS 2.0, the Web Content Integrator provides a loosely-coupled mechanism for transferring published content and meta data to the portal after they have been approved in the source system. Once the content and meta data have been transferred to the portal, it is possible to leverage the built-in content management features of WebSphere Portal to secure, personalize, and display the content to end users.

Feed Service System Design





Feed Service Facilitates persistence of configuration data and execution of feed processing.

Feed Admin Servlet Manage task schedules and trigger feed processing via URL commands

Admin Portlets provide a UI for creating and configuring Feed Job and Task definitions. Also allows manual triggering of feed processing.

WAS Scheduler performs execution of tasks on scheduled intervals.

Feed Service EJB is called by the WAS scheduler when a scheduled task is fired. Creates an execution thread and calls the Feed Service to locates the proper Job and initiate feed processing.

Domain Extensions are triggered by the Feed Service to perform feed processing. These are located by the Extension Point Registry. The Web Content Integrator is a domain extension that connects to WCM.

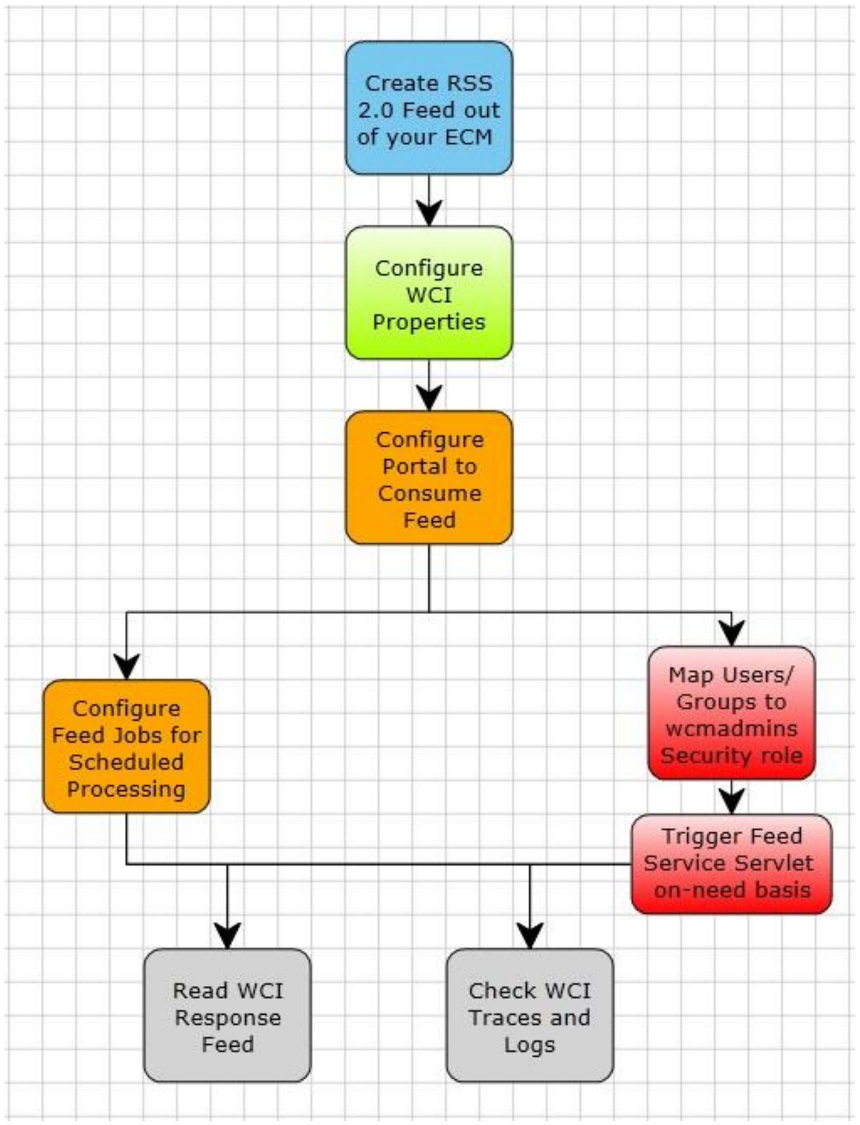
Content Feeds, can be located anywhere as long as they can be accessed by the Feed Service over HTTP.

Feed Task Registry keeps feed location info, scheduler task ID, feed/handler bindings, and domain-specific properties.

Object ID Mapping is an service which allows the domain extensions to store mappings between the internal and external IDs for a given content object.

Job & Feed Configurations are created via the Feed Service and registered with the scheduler to be executed. Tasks typically process the feed by calling a domain extension plug-in.

Using Web Content Integrator


Create RSS 2.0 Feed out of your ECM
Configure WCI Properties
Configure Portal to Consume Feed
Configure Feed Jobs for Scheduled Processing
Map Users/ Groups to wcmadmins Security role
Trigger Feed Service Servlet on-need basis
Read WCI Response Feed
Check WCI Traces and Logs

Input and Output Feeds

See sample feed files as additional attachments with this article.

The external ECM system should create RSS 2.0 feed format to impose WCI with. A sample Input feed that you can use as a reference to understand about the structure and format. It is flooded with comments and guidance in order for you to build more reliable XSD file that covers all the options for the feed formation.

The sample input feed is a well-formed XML file focuses mainly in feeding Content item, with comments showing the difference if feeding Site Area, Category, or Component.

I advise, NOT to automatically create the XSD file using, maybe RSA based on the feed sample, simply because each element might have too many options that it is only mentioned in the comment sections...etc. Please read carefully the comments, in order to amend the resulting XSD file accordingly.

There are some special handling, the producer should be aware of and pay a good attention to, while producing the feed files. For example:

- Rich Text or HTML element could have embedded link to images, files, or other contents. These embedded links are represented in tags (<img src=""></img> or <a href=""></a>); those tags need to be replaced, with <link type=""> element, for the WCI to pick up the referred file/image/content from within the feed, and create a relevant reference links (with WCM tags) to put within the Rich Text or HTML element.
    - To refer to another content within the feed, embedded link in Rich Text or HTML elements, must be replaced with <link type="content" guid="UniqueID-of-The-Content-In-The-Feed""/>
    - To refer to a file within the feed, embedded link in Rich Text or HTML elements, must be replaced with <link type="file" guid="UniqueID-of-The-File-Component-In-The-Feed"/>
    - To refer to an image within the feed, embedded link in Rich Text or HTML elements, must be replaced with <link type="image" guid="UniqueID-of-The-Image-Component-In-The-Feed""/>
- Handshaking feature between producer and WCI is important, even if the transfer is only in one direction. Handshaking makes it easy for the producer to know what exactly has been processed and what it should bring on in the next feeds. It also helps WCI not to try to process anything, if none has been modified. Handshaking protocol could be implemented either in the HTTP headers or within the feed itself. While our producer more likely to drop a feed file somewhere and provide the URL to WCI, it's possibly unlikely for the producer to manipulate with the HTTP headers, thus we are going to use Handshaking protocol within the feed, by:
    - Providing the <lastBuildDate> with what represent the recent update to the feed
    - Providing the <ibmfs:etag> Entity Tag with a label that represent this feed instance.

  WCI will always save the upcoming handshaking parameters after it successfully processes the feed, and it ships it back as query strings to the producer in the next time while asking for the next feed, for the later to know what has been successfully processed lately!

  Ideally, if no changes have occurred since the last time WCI requested the feed, the producer should respond with an HTTP 304 (Not Modified) response code. This will signal the consumer that there are no changes and thus that it does not need to attempt to parse the feed and do the subsequent processing. Otherwise, WCI will try to parse the existing file anyway, in our case; thus, I strongly recommend that the feed file to be dropped and controlled with (HTTP Sever) , where you can build some rules for the server to respond to WCI with HTTP 304 (Not Modified) if that file has not been replaced recently!

- If you are only using Feed Service Servlet, then you intentionally request WCI to process your feeds on-need-basis. Handshaking protocol won't add too much value with this approach.

Input feeds can be located anywhere, as long as they can be accessed by WCI through HTTP.

Additionally, WCI can also produces an "Output" feed to the producer with the processing end results for each item. I have also created a sample for the output feed. Normally WCI responds with an Output feed only when it's triggered through its Feed Service Servlet,. Feed Service Servlet provides HTTP Servlet Response with mime type of (application/xml) with a character set of (UTF-8),. The response format is documented in the product wiki >> Please read more about Results Feed

We can force WCI to create a response feed all the time, even when it is triggered by Feed Jobs. This is explained later in configuring feed properties.

Feed Configuration Properties

Feed configuration properties file, named (WCMConsumerPlugins.properties) is a properties file where you can adjust some WCI global attributes. The file is located in

```
wp_profile/PortalServer/wcm/shared/app/config/WCMConsumerPlugin.properties
```

Below we are stressing on a some of the important attributes out of that file.

1. There is an option in WCI to save the producer from match/find/replace the embedded links in (HTML or Rich Text elements) that refer to images, as WCI can be configured to automatically sniff for the <img src=""/> tags, upload them as library image components, and then rewrite the correct references in the element. This feature is disabled by default in WCI. If you would like to use it, the following precautions are important:

   - Image references matching the pattern “<img src= .." are processed. Image references that are included within Javascript code or CSS styles will not be processed.

   - URLs specified in the "src" attributes of those image tags are converted to fully qualified URLs using other information about the content item in the feed. Relative links must be relative to the URL in the <link> element of the feed item following the standard rules for relative links. I advise you provide a fully qualified URL for images up-front.

   - Query strings in the src=... URL that possible would have some parameters, will be ignored.

To enable this feature, you should set the below property with TRUE

```
disable.img.proc=false
```

2. If your external ECM system is only using Feed Job and has access to WCI file system. You can force WCI to always respond with Output feed discussed earlier. Enable this feature and specify the desired response file location using the below properties

```
# Support to write an output feed to a file. By default enable.output.feed is false.

enable.output.feed=false

# The path where the output feed file will be created. A file named output_<job_id>.xml is created in the specified path. By default the path is <wps_profile>/installedApps/<node_name>/FeedService.ear if it is left empty.

output.feed.path=
```

See a copy of the feed properties file additional attachment with this article.

Using Feed Administration Portlets

Portal has two administrative Portlets related to WCI, "WCI Feeds", and "Feed Jobs".

Regardless of you are using feed jobs or only Feed Servlet, WCI Feeds portlet, located in Portal Administration >> Portal Content >> Feed Configuration, is where you define your feed meta data i.e.

- Feed Name
- Feed URL (Must be HTTP)
- Credential Slot for Feed (if your Feed location mandates a username/password)
- Web Content Library (The default WCM Library that WCI processing with work against)
- Credential Slot for Library (To understand the role authorization that can be work on the library)
- Pass Handshake Data (either in HTTP Header or Feed). Please read more about Handshaking >> Handshake Protocol

"Feed Jobs" portlet, located in Portal Administration >> Portal Content >> Feed Jobs, is used when you plan to periodically trigger WCI feed service, not on-need-basis (in latter case, you better use Feed Servlet). In this portlet you may:

- Define a Job name.
- Frequency of the Job processing
- Define specific time to run the Job
- Group the Feeds to be affected by this job. (Feeds are already created in "WCI Feeds" portlet)
- Run, Suspend, Reset...etc. Job

You can also control configured Jobs using the Feed Servlet.

Feed Service Servlet

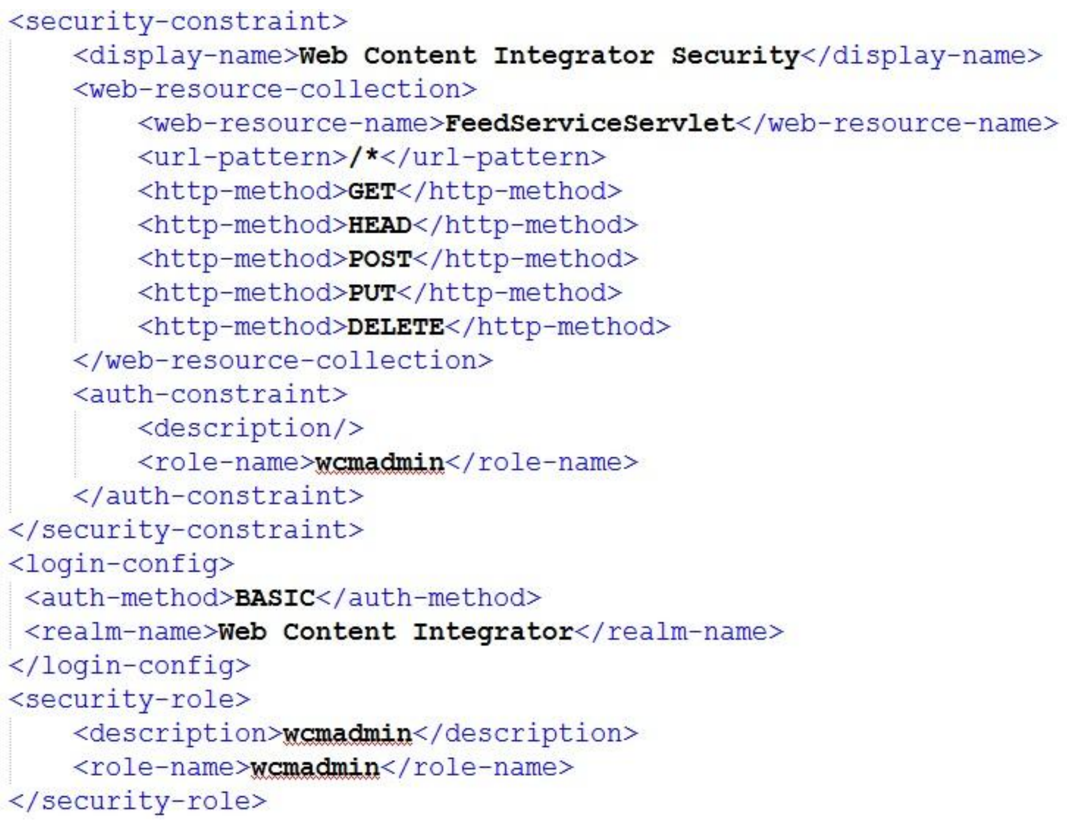
Normally if you plan to use the Feed Service Servlet, then you may be have the following justification:

- Your updates involve bulk content items consumption in batches with unknown duration between each batch (could be hours/days/weeks).
- You want to control when the processing to start, to avoid conflicts.
- You do not have access to WCI file system to read WCI Output response feed.

Feed Service Servlet is coming out-of-the-box with WCI EAR package. It is installed while WCI is installed and it is ready to be used. The Servlet is very powerful that gives you a set of important controls and functions. It can control Feed Jobs or Single Feed URLs configured through administration.

The default URI of the Servlet is (/wps/Feed/FeedServiceServlet).

And because the Servlet gives you access to a lot of functionality, its web.xml has the following lines



```
<security-constraint>
    <display-name>Web Content Integrator Security</display-name>
    <web-resource-collection>
        <web-resource-name>FeedServiceServlet</web-resource-name>
        <url-pattern>/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>HEAD</http-method>
        <http-method>POST</http-method>
        <http-method>PUT</http-method>
        <http-method>DELETE</http-method>
    </web-resource-collection>
    <auth-constraint>
        <description/>
        <role-name>wcmadmin</role-name>
    </auth-constraint>
</security-constraint>
<login-config>
 <auth-method>BASIC</auth-method>
 <realm-name>Web Content Integrator</realm-name>
</login-config>
<security-role>
    <description>wcmadmin</description>
    <role-name>wcmadmin</role-name>
</security-role>
```

Meaning is the servlet is secured with BASIC HTTP Authentication and only users/groups assigned to the security role (wcmadmin) can access it. So, before you can use the servlet, you MUST map users/groups to wcmadmin role through WAS administration in this location (Applications >> Application types > >

WebSphere Enterprise Applications >> WCI). Under Detail Properties, click Security role to user/group mapping.

You may delete these parts from web.xml and redeploy the Servlet WAR, so you avoid providing credential while calling the servlet or you assign "Everyone" to this role, but both approaches are NOT recommended at all.

The recommendation is to create a Portal user i.e. "FeedCaller" annd map that user to wcmadmin security role of the servlet.

The calling modules (clients) MUST provide the credential of that user through BASIC HTTP Authentication, each time they make a call to the feed servlet. They must which use a request header to store the username and password in Base64 encoding. Clients may leverage some libraries i.e. Apache's HttpClient.

Since this is BASIC authentication, the password will be passed to the servlet in plain text. This is not recommended, hence, it is advised to make the call over SSL.

The Servlet has the following parameters and it will respond back with WCI Output response feed discussed earlier in the HTTP Response. The parameters of the servlet are:

action	This is mandatory. It expects one of the following actions >> (runJob, stopJob, consumeFeed, clearCache, resetFeed, deleteImportedItems, or listPlugins)
feed	This is mandatory that expects the feed name, if you select "consumeFeed", "clearCache", "resetFeed", or "deleteImportedItems" actions. It refers to the feed name provided in Portal Administration >> Portal Content >> Feed Configuration page.
url	Expects corresponding feed URL. This is optional if you select "consumeFeed", "clearCache", "resetFeed", or "deleteImportedItems" actions. If not provided, it will pick it up from Portal Administration >> Portal Content >> Feed Configuration page.
job	This is mandatory that expects Job name, if you select "runJob" or "stopJob" actions. A job configured has a feed list (several feeds sorted), if we select any of these actions, it will affect all feeds included in that job. Jobs are created through Portal Administration >> Portal Content >> Feed Jobs page.

For example, if you are planning to use the servlet only to control individual feeds (rather than Jobs), the scenario would be as follows:

1. Since external modules will control the consumption process, we do not need any Feed scheduled Jobs.
2. Through Portal Administration >> Portal Content >> Feed Configuration page, we create a single feed entry. We provide a unique name and its expected HTTP URL (preferably placed in an HTTP Server)

3. Client component places/overwrites the feed file in the location provided in step 2. Feed itself should include the Entity Tag and Last Modified Date elements required for handshaking. We will are not using handshaking via HTTP headers.
4. Client component calls out WCI Feed Service Servlet with the following query string (?action=consumeFeed&feed=FEED_NAME_AS_CREATED_IN_ADMINISTRATION)
5. Feed Service Servlet provides HTTP Servlet Response with mime type of (application/xml) with a character set of (UTF-8).
6. External components should leverage the use of Abstract Factory and Observer Design patterns for smoother transitions/iterations.
7. If it uses Observer design pattern, clients would have a notification mechanism to be informed when a response is ready.
8. Client Component parses WCI response and understands what went wrong for each item in the feed.
9. Client component may consider other Feed Servlet actions, in a subsequent recall, i.e. resetFeed, or deleteImportedItems...etc. if the first results were catastrophic! It may keep doing this, until the response is accepted and can be tolerated.
10. If everything go smooth, or errors can be tolerated in sub-sequent feeds, Client component repeats the process, starting step 3.

Other Content Acquisition Approaches

WebDAV

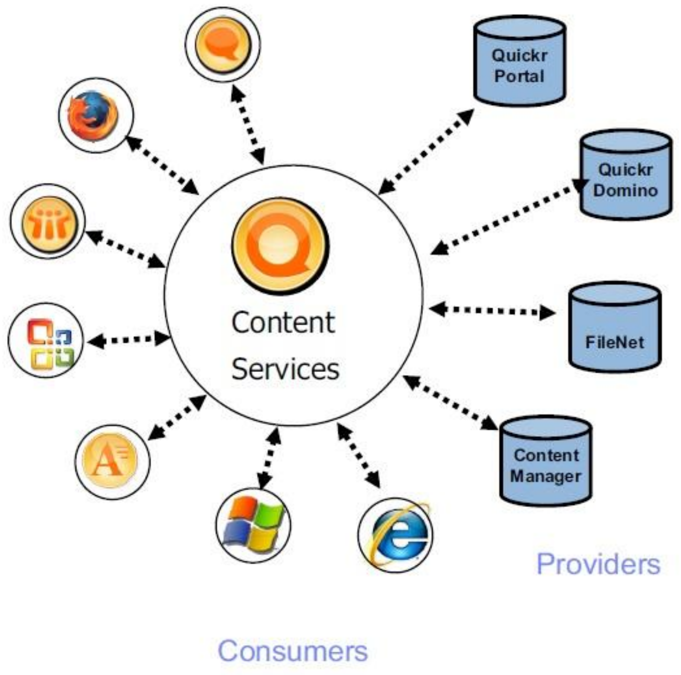
WebDAV is a set of extensions to the HTTP protocol that enables users to collaboratively edit and manage files on remote web servers; in IBM WCM, it can be used to expose WCM repository. WebDAV enables authenticated users to create, modify, and delete items in WCM . It currently supports the following items:

- Libraries
- Components
- Taxonomies and Categories
- Site Areas
- Presentation Templates.

A common use of WebDAV is to show WCM repository as a file system in external tools, such as HTML editors (for example, Dreamweaver). ECM systems can use WebDAV API to push documents, that are most likely files, into WCM.

Content Management Interoperability Services (CMIS)

The following block illustrates the essence of Content On-Demand



In WCM, the federated documents feature enables you to retrieve information about documents in a remote repository and insert links to those documents in your web content. Several types of feeds are supported for accessing remote content. The federated documents feature can access content from remote systems that support the following methods:

- Content Management Interoperability Services 1.0 (CMIS 1.0)
- Document Services remote interfaces, as supported by Lotus Quickr, IBM Content Manager, and FileNet Content Manager
- Atom feeds (Personalization rules only)

WCM REST Services

Application developers can use Representational State Transfer (REST) services to work with Web Content Manager. The REST service for Web Content Manager provides authoring access to content items and elements. The service follows the Atom Publication Protocol, and atom feeds, and entries are accessible in XML (application/atom+xml) and JSON (application/json) format.

AtomPub standard is well suited to content creation. Compared to WCI, instead of providing a feed consumer, clients POST individual content items. In practice this might, in some cases, make it easier for clients since they receive an individual response for each request. While the WCI concept allows for forward referencing in the feed, there are a lot of behaviors in the feed consumer which need to be understood before it can be used effectively.

There are XSD files shipped with WCM which accurately describe the XML format for each item, and unlike the WCI which has lots of un-typed attributes which cannot be effectively validated, the XSDs provide strong typing for all properties. When 3rd party feeds contain errors it will be a lot more obvious what the error is.

The security model of REST is much more sophisticated. There are a number of security roles which can be configured to allow 3rd parties to execute defined queries against the content repository, as well as controlling who can see or create content.

REST is the primary integration technology for WCM moving forward. It already underpins a number of our recent capabilities, notably the in-place editing features introduced in 8001.

REST Service or WCI

You may eventual favor WCI over REST services.. Maybe for the following thoughts and according to your client's specific requirements:

- WCM REST Services approach feels more into replacing/decreasing the need of using WCM APIs or server side components (i.e. JSP components) to process WCM repository, especially if this processing is more likely minor, frequent, and over-the-air (i.e. In-line editing, feeding stock/news information...etc.). On the other hand, WCI feels more part of Portal content acquisition avenues alongside with CIMS, WebDAV, and Federated Contents that aims to make WCM coexists with other similar ECM systems working together.
- Your use cases may involve bulk content items consumption in batches with unknown duration between each batch (could be hours/days/weeks). This is notably done with a single feed for WCI on-need-basis, but this implies successive POST requests for every teeny tiny item/element/component within the batch in case we use REST Service.
- WCI supports automatic processing of embedded images within HTML and Rich Text elements. REST Services would need to handle each image instance as an dedication POST attachment request, with binaries and all
- Same goes for embedded files and referred contents. Indeed WCI does not automatically process those, but the preprocessing required by the clients is as simple as some tag replacements
- WCI supports forward referencing for items inside the feed, or items already processed by WCI in previous feeds consumption.
- WCI gives the producer the flexibility to create its own custom (possibly logically built) GUIDs for each item in the feed, and also maintains these custom GUIDs in WCM repository together with its counterpart WCM UUIDs created while processing. With REST Services, you need to know/fetch the long complex WCM UUIDs of items you target in each POST as appropriate.

- WCI keeps track of all the changes a feed has made since it is first processed; at any point in time, a producer can execute "Restore Factory Settings"-like action (RESET), and every change that have been made will be flushed away. Similar actions like deleting imported items...etc. are also useful.

Conclusion

You have learned when to WCI as an option for the content acquisition process. You also learned when to favor the use WCI Feed Service Servlet, aside from using the administration portlets only. You learned how to configure WCI feed service properties and why you should assign security role to WCI Servlet. You know about the Servlet URL and its required parameters and the expected response feed.