GST Site Foundation Developer's Guide v1.0

Last Updated: January 28, 2011

Contributors: Tony Field
Raman Gill
Dolf Dijkstra

Table of Contents

1 PART I: INSTALLATION

1.1 Introduction

This is a complete guide to install the GST Site Foundation onto FatWire Content Server.

Some steps may go into more detail for many users but it is designed so a novice user of FatWire Content Server can integrate the GSF with minimal external resources required. Intermediate and advance users can obtain the required configuration settings from the first few lines of each heading and move onto the next step.

The environment used in the samples is Mac OS X, FatWire Content Server 7.5 Patch 3, bash, Apache Tomcat 6.0.18 (with CS), and Apache Maven 2.2.0, Java 1.6.

This guide will display the directory location of installed components; by no means do your directories have to match the ones outlined here. These directories are explicitly mentioned to aid some users through this process.

1.2 Further Reading

GST Foundation Website Link: http://gsf.fatwire.com/

GST Foundation Minimal Installation Guide Link: http://www.nl.fatwire.com/dta/contrib/gst-foundation/InstallGuide.html

GST Foundation ContentServer Servlet Profiling Tools: http://www.nl.fatwire.com/dta/contrib/webapp-profiling/

Subversion (version control system): http://svnbook.read-bean.com/

Maven (software project management & comprehension tool): http://maven.apache.org/

Tuckey URL Rewrite: http://www.tuckey.org/urlrewrite/

1.3 Install and configure Content Server

This guide uses Content Server (CS) 7.5 Patch 4; make sure you set it to use Apache Tomcat. For example I have it installed in: `/Users/rgill/Library/JSK_CS753_CLEAN`

Note: I enabled all the Sample Sites. This is my sample and sandbox Jump Start Kit (JSK).

The following are configurations are required for Content Server and the Application Server to utilize the GSF.

i) Edit `your futuretense_xcel.ini` file located in your `JSK/ContentServer/version#` folder
For example: `/Users/rgill/Library/JSK_CS753_CLEAN/ContentServer/7.5.3`
Set the following property:
`xcelerate.pageref=com.fatwire.gst.foundation.url.WraPageReference`

Note: use # if you wish to comment the existing entry and the add this entry under it

For example:

```
#xcelerate.pageref=com.openmarket.xcelerate.publish.PageRef
xcelerate.pageref=com.fatwire.gst.foundation.url.WraPageReference
```

ii) Edit each CS environment set the system property `com.fatwire.gst.foundation.env-name` in your `catalina.bat` or `catalina.sh` (assuming you're using Apache Tomcat) located in your `Application Server/bin`.
You can set `com.fatwire.gst.foundation.env-name=fatwire-dev` at the end of:
`set CATALINA_OPTS=%CATALINA_OPTS%` [catalina.bat] or `CATALINA_OPTS="$CATALINA_OPTS` [catalina.sh]
For example: `/Users/rgill/Library/JSK_CS753_CLEAN/App_Server/apache-tomcat-6.0.18/bin/catalina.sh` has the following

```
CATALINA_OPTS="$CATALINA_OPTS -Xms384m -Xmx512m -XX:MaxPermSize=128m -Dfile.encoding=UTF-8 -
Dcom.fatwire.gst.foundation.env-name=fatwire-dev"
```

1.4 Import Content Server jars into Maven

Ensure that Maven is installed. For details refer to Appendix A.

We need to add the Content Server jars to Maven. You can download a shell script that automates this process. If you're on Windows you can rename the file to have the extension to .bat and change the file to conform to Command Prompt commands. You can also just copy and paste the commands to your terminal / command prompt. Note: You can configure the script to run anywhere as long as you provide the proper path to the jars in the import script.

i) Download the script file (or copy from next page) to your `Application Server/webapps/cs/WEB-INF/lib` directory.
My path would be: `/Users/rgill/Library/JSK_CS753_CLEAN/App_Server/apache-tomcat-6.0.18/webapps/cs/WEB-INF/lib`
Download from: http://source.developernet.fatwire.com/devnet/contrib/ (DOS batch format, convert to shell script if required).

ii) Edit the `mvn-import.sh` file to add the correct `VERSION`. ex: `VERSION=7.5.3`

iii) Edit the file so any jar that is not in your `WEB-INF/lib` directory is not imported.
For example, in my CS 7.5 P3, I do not have the following jars: `commercedata.jar`, `cs-portlet.jar`, and `spark.jar`
These have been commented out, comments in Windows are REM and in *NIX its #.

iv) Change into the directory of your `WEB-INF/lib`. Either run the script or copy the commands to your terminal / prompt.
These jars will install into your Maven and are mandatory.

```
#!/bin/sh
VERSION=7.5.3
mvn -B install:install-file -Dfile=assetapi-impl.jar -DgroupId=com.fatwire.cs -DartifactId=assetapi-impl -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=assetapi.jar -DgroupId=com.fatwire.cs -DartifactId=assetapi -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=assetframework.jar -DgroupId=com.fatwire.cs -DartifactId=assetframework -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=assetmaker.jar -DgroupId=com.fatwire.cs -DartifactId=assetmaker -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=basic.jar -DgroupId=com.fatwire.cs -DartifactId=basic -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=batch.jar -DgroupId=com.fatwire.cs -DartifactId=batch -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=catalog.jar -DgroupId=com.fatwire.cs -DartifactId=catalog -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
#mvn -B install:install-file -Dfile=commercedata.jar -DgroupId=com.fatwire.cs -DartifactId=commercedata -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=cs-core.jar -DgroupId=com.fatwire.cs -DartifactId=cs-core -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
#mvn -B install:install-file -Dfile=cs-portlet.jar -DgroupId=com.fatwire.cs -DartifactId=cs-portlet -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=cs.jar -DgroupId=com.fatwire.cs -DartifactId=cs -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=cscommerce.jar -DgroupId=com.fatwire.cs -DartifactId=cscommerce -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=directory.jar -DgroupId=com.fatwire.cs -DartifactId=directory -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=firstsite-filter.jar -DgroupId=com.fatwire.cs -DartifactId=firstsite-filter -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=firstsite-uri.jar -DgroupId=com.fatwire.cs -DartifactId=firstsite-uri -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=flame.jar -DgroupId=com.fatwire.cs -DartifactId=flame -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=framework.jar -DgroupId=com.fatwire.cs -DartifactId=framework -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=gator.jar -DgroupId=com.fatwire.cs -DartifactId=gator -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=gatorbulk.jar -DgroupId=com.fatwire.cs -DartifactId=gatorbulk -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=ics.jar -DgroupId=com.fatwire.cs -DartifactId=ics -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=logging.jar -DgroupId=com.fatwire.cs -DartifactId=logging -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=lucene-search.jar -DgroupId=com.fatwire.cs -DartifactId=lucene-search -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=MSXML.jar -DgroupId=com.fatwire.cs -DartifactId=MSXML -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=rules.jar -DgroupId=com.fatwire.cs -DartifactId=rules -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=sampleasset.jar -DgroupId=com.fatwire.cs -DartifactId=sampleasset -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
#mvn -B install:install-file -Dfile=spark.jar -DgroupId=com.fatwire.cs -DartifactId=spark -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=sparksample.jar -DgroupId=com.fatwire.cs -DartifactId=sparksample -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=sseed.jar -DgroupId=com.fatwire.cs -DartifactId=sseed -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=sserve.jar -DgroupId=com.fatwire.cs -DartifactId=sserve -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=transformer.jar -DgroupId=com.fatwire.cs -DartifactId=transformer -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=visitor.jar -DgroupId=com.fatwire.cs -DartifactId=visitor -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
mvn -B install:install-file -Dfile=xcelerate.jar -DgroupId=com.fatwire.cs -DartifactId=xcelerate -Dversion=$VERSION -Dpackaging=jar -DgeneratePom=true
```

1.5 Obtain the GSF

It is necessary to obtain the GSF from the Internet. The bundle is comprised of several types of components, including a jar file, several database table files, a tag library, documentation, etc.

The Jar file can be downloaded directly from the following URL:

http://www.nl.fatwire.com/maven2/com/fatwire/gst/gst-foundation-all/1.0/

However, to obtain the other resources subversion must be used. It should be noted that only the Jar file requires compilation. Therefore it is not necessary to follow through all of the build steps immediately below if you just plan to download the completed jar.

1.5.1 Download GSF using Subversion

We now need to obtain the source files for the project. We use the svn command to retrieve these files.

Run the command: `svn export http://www.nl.fatwire.com/svn/dta/contrib/gst-foundation/tags/gsf-parent-1.0 gst-foundation`

Example:

```
$ mkdir gsf
$ cd gsf
$ svn export http://www.nl.fatwire.com/svn/dta/contrib/gst-foundation/tags/gsf-parent-1.0 gst-foundation
```

The output to the screen will be the files retrieved from the source repository; it will appear similar to the following entries:

```
A    gst-foundation/gsf-cs-test
A    gst-foundation/gsf-cs-test/src
A    gst-foundation/gsf-cs-test/src/main
A    gst-foundation/gsf-cs-test/src/main/java
A    gst-foundation/gsf-cs-test/src/main/java/com
A    gst-foundation/gsf-cs-test/src/main/java/com/fatwire
A    gst-foundation/gsf-cs-test/src/main/java/com/fatwire/gst
A    gst-foundation/gsf-cs-test/src/main/java/com/fatwire/gst/foundation
A    gst-foundation/gsf-cs-test/src/main/java/com/fatwire/gst/foundation/test
A    gst-foundation/gsf-cs-test/src/main/java/com/fatwire/gst/foundation/test/jndi
A    gst-foundation/gsf-cs-test/src/main/java/com/fatwire/gst/foundation/test/jndi/VerySimpleInitialContextFactory.java
A    gst-foundation/gsf-cs-test/src/main/java/com/fatwire/gst/foundation/test/CSTest.java
A    gst-foundation/gsf-cs-test/src/main/resources
A    gst-foundation/gsf-cs-test/src/main/resources/satellite.properties
A    gst-foundation/gsf-cs-test/src/main/resources/futuretense.ini.tmpl
A    gst-foundation/gsf-cs-test/src/main/resources/datasource.properties
A    gst-foundation/gsf-cs-test/src/main/resources/ServletRequest.properties
A    gst-foundation/gsf-cs-test/src/main/resources/log4j.xml
A    gst-foundation/gsf-cs-test/pom.xml
A    gst-foundation/gsf-taglib
… (Many more files)
```

It will save the source files in a directory called gst-foundation.

For example I run the svn command in `/Users/rgill/Library/gsf`

This will create a directory `/Users/rgill/Library/gsf/gst-foundation`

The contents in the directory should be similar to:

```
drwxr-xr-x+ 1 rgill None         0 2010-09-01 08:46 gsf-assembly
drwxr-xr-x+ 1 rgill None         0 2010-09-01 08:46 gsf-cs-all
drwxr-xr-x+ 1 rgill None         0 2010-09-01 08:46 gsf-cs-test
drwxr-xr-x+ 1 rgill None         0 2010-09-01 08:46 gsf-facades
drwxr-xr-x+ 1 rgill None         0 2010-09-01 08:46 gsf-tagging
drwxr-xr-x+ 1 rgill None         0 2010-09-01 08:46 gsf-taglib
drwxr-xr-x+ 1 rgill None         0 2010-09-01 08:46 gsf-wra
-rw-r--r--+ 1    rgill None 2725 2010-09-01 08:46 pom.xml
drwxr-xr-x+ 1 rgill None         0 2010-09-01 08:46 src
```

1.5.2 Build the Maven project

We will now run the Maven command to clean the working folder and install the files into the repository.

We run the Maven command on the gst-foundation `pom.xml` file, located in the gst-foundation directory created from the previous step.

Within the `gst-foundation` directory we run the `mvn clean install` command.

For example: in `/Users/rgill/Library/gsf/gst-foundation`

Run the command: `mvn clean install`

Note: If the `mvn` command does not work make sure you properly installed and configured Maven on your system. View Appendix A for more details.

On the first run, many dependencies will automatically be downloaded and installed. This is normal, and one of the primary benefits of Maven.

The output would be something like:

```
oberon:gst-foundation tfield$ mvn clean install
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   FatWire GST: Site Foundation
[INFO]   FatWire GST: Site Foundation ContentServer jars
[INFO]   FatWire GST: Site Foundation ContentServer Library Facades
[INFO]   FatWire GST: Site Foundation Web-Referenceable Asset Support
[INFO]   FatWire GST: Site Foundation Tagging support
[INFO]   FatWire GST: Site Foundation Taglibs
[INFO]   FatWire GST: Site Foundation Aggregator
[INFO] ------------------------------------------------------------------------
[INFO] Building FatWire GST: Site Foundation
[INFO]    task-segment: [clean, install]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
…
[INFO] [install:install {execution: default-install}]
[INFO] Installing /Users/tfield/tmp/gst-foundation/gsf-assembly/target/gst-foundation-all-1.0.jar to /Users/tfield/.m2/repository/com/fatwire/gst/gst-foundation-
all/1.0/gst-foundation-all-1.0.jar
[INFO] Installing /Users/tfield/tmp/gst-foundation/gsf-assembly/target/gst-foundation-all-1.0-sources.jar to /Users/tfield/.m2/repository/com/fatwire/gst/gst-foundation-
all/1.0/gst-foundation-all-1.0-sources.jar
[INFO] Installing /Users/tfield/tmp/gst-foundation/gsf-assembly/target/gst-foundation-all-1.0-sources.jar to /Users/tfield/.m2/repository/com/fatwire/gst/gst-foundation-
all/1.0/gst-foundation-all-1.0-sources.jar
[INFO]
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO] ------------------------------------------------------------------------
[INFO] FatWire GST: Site Foundation ........................... SUCCESS [10.798s]
[INFO] FatWire GST: Site Foundation ContentServer jars ....... SUCCESS [4.142s]
[INFO] FatWire GST: Site Foundation ContentServer Library Facades  SUCCESS [53.430s]
[INFO] FatWire GST: Site Foundation Web-Referenceable Asset Support  SUCCESS [3.859s]
[INFO] FatWire GST: Site Foundation Tagging support .......... SUCCESS [3.497s]
[INFO] FatWire GST: Site Foundation Taglibs .................. SUCCESS [3.492s]
[INFO] FatWire GST: Site Foundation Aggregator ............... SUCCESS [0.584s]
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1 minute 20 seconds
[INFO] Finished at: Thu Nov 25 21:33:39 EST 2010
[INFO] Final Memory: 63M/379M
[INFO] ------------------------------------------------------------------------
oberon:gst-foundation tfield$
```

1.5.3 Compile the Java and build the jar

While the jar was built in the previous chapter, it is in a location that is inconvenient. Re-build the jar using the command: `mvn clean package`.

Example:

```
oberon:gst-foundation tfield$ mvn clean package
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   FatWire GST: Site Foundation
[INFO]   FatWire GST: Site Foundation ContentServer jars
[INFO]   FatWire GST: Site Foundation ContentServer Library Facades
[INFO]   FatWire GST: Site Foundation Web-Referenceable Asset Support
[INFO]   FatWire GST: Site Foundation Tagging support
[INFO]   FatWire GST: Site Foundation Taglibs
[INFO]   FatWire GST: Site Foundation Aggregator
[INFO] ------------------------------------------------------------------------
[INFO] Building FatWire GST: Site Foundation
[INFO]    task-segment: [clean, package]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] [site:attach-descriptor {execution: default-attach-descriptor}]
[INFO] [javadoc:jar {execution: attach-javadocs}]
[INFO] Not executing Javadoc as the project is not a Java classpath-capable package
[INFO] ------------------------------------------------------------------------
[INFO] Building FatWire GST: Site Foundation ContentServer jars
[INFO]    task-segment: [clean, package]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] [site:attach-descriptor {execution: default-attach-descriptor}]
Downloading: http://www.nl.fatwire.com/maven2/com/fatwire/cs/cs/7.5.3/cs-7.5.3.pom
…
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing /Users/tfield/tmp/gst-foundation/gsf-assembly/target/gst-foundation-all-1.0.jar with /Users/tfield/tmp/gst-foundation/gsf-assembly/target/gst-foundation-
all-1.0-shaded.jar
[INFO] Replacing /Users/tfield/tmp/gst-foundation/gsf-assembly/target/gst-foundation-all-1.0-sources.jar with /Users/tfield/tmp/gst-foundation/gsf-assembly/target/gst-
foundation-all-1.0-shaded-sources.jar
[INFO]
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO] ------------------------------------------------------------------------
[INFO] FatWire GST: Site Foundation ......................... SUCCESS [8.010s]
[INFO] FatWire GST: Site Foundation ContentServer jars ..... SUCCESS [5.096s]
[INFO] FatWire GST: Site Foundation ContentServer Library Facades  SUCCESS [1:02.093s]
[INFO] FatWire GST: Site Foundation Web-Referenceable Asset Support  SUCCESS [4.054s]
[INFO] FatWire GST: Site Foundation Tagging support ......... SUCCESS [3.263s]
[INFO] FatWire GST: Site Foundation Taglibs ................. SUCCESS [3.649s]
[INFO] FatWire GST: Site Foundation Aggregator .............. SUCCESS [3.298s]
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

```
[INFO] Total time: 1 minute 30 seconds
[INFO] Finished at: Thu Nov 25 21:22:34 EST 2010
[INFO] Final Memory: 62M/379M
[INFO] ------------------------------------------------------------------------
oberon:gst-foundation tfield$
```

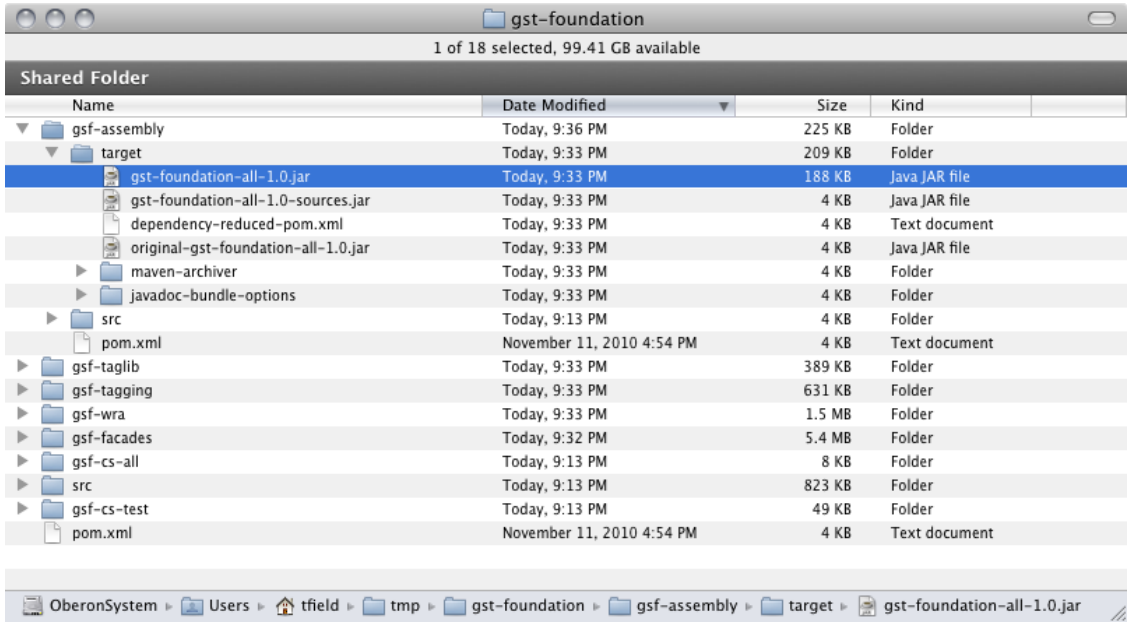
1.6 Add GSF jar to Content Server

Copy the `gst-foundation-all-1.0.jar` from the working directory to your `WEB-INF/lib` directory. If your Jump Start Kit is running, shut it down, and copy the file. Then restart the JSK.

Note: the Build must be successful in the previous steps in order for the jar file to be present. You can try to do a hot-swap with the jar file but it is preferable to shut down ContentServer and restart.

For example. `Copied: /Users/rgill/Documents/gst/gst-foundation/gsf-assembly/target/gst-foundation-all-1.0.jar`

`To: /Users/rgill/Library/JSK_CS753_CLEAN/App_Server/apache-tomcat-6.0.18/webapps/cs/WEB-INF/lib`



1.7 Add Status Servlet Filter

This filter allows Content Server to set HTTP status code headers. It's used in conjunction with the controller to return 404, 403, and other http status codes in a normal way.

To enable the filter you need to modify the web.xml file.

Add to your web.xml located in your Application Servers' Context's WEB-INF directory the required filter settings. Please see the Status servlet filter section from the webapp-profiling site.

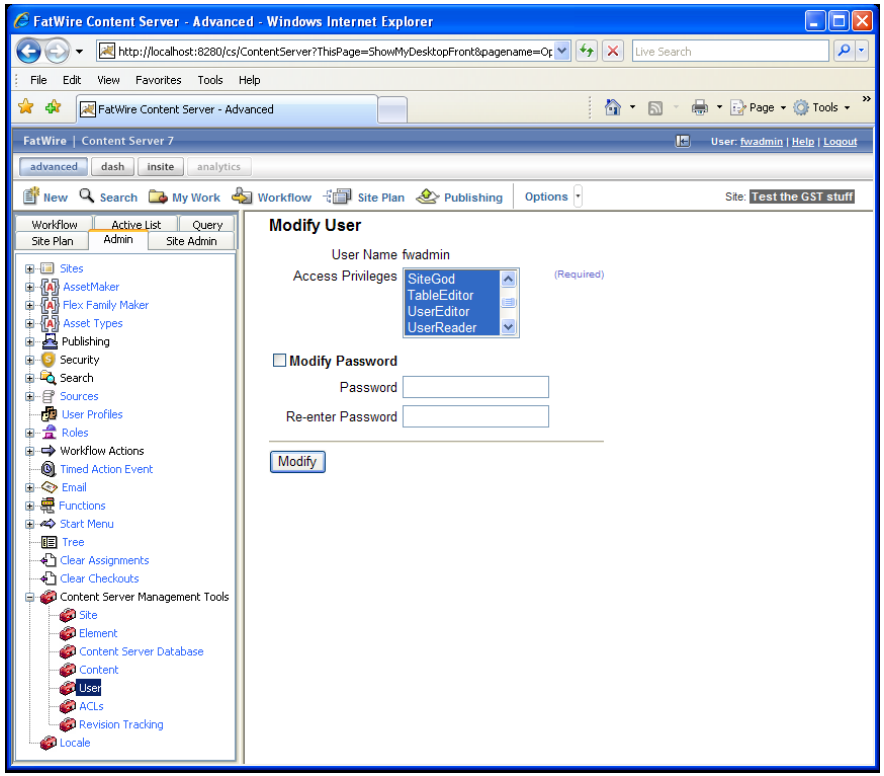
```
<filter>
  <filter-name>CustomHeaderFilter</filter-name>
  <filter-class>com.fatwire.gst.foundation.httpstatus.CustomHeaderFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>CustomHeaderFilter</filter-name>
  <url-pattern>/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

1.8 Log in to Content Server with Appropriate Privileges

Log into Content Server as a user who has SiteGod and xceladmin privileges. Specific scripts need to be executed in an upcoming section. To successfully execute these scripts, your user must have the SiteGod and xceladmin ACLs assigned.

For example if your user account is fwadmin. In CS -> Admin (tab) -> Content Server Management Tools (expand item) -> User (item) -> Enter User Name (textbox), choose Modify User (radio selection) -> OK (button) -> select user (link) -> Add Access Privilege 'SiteGod' (multi-select item) -> Modify (button)

You should logout and re-login.

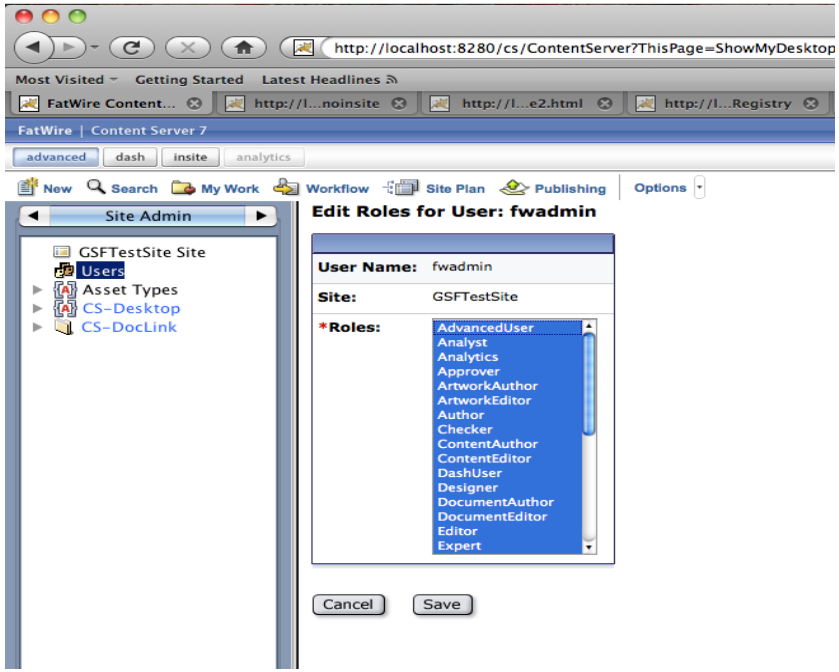


1.9 Create a New Site

Create a new site, for example GSFTestSite, make sure you have added your user from the previous section. Also make sure they have the appropriate Roles for the new site you created.

Create a site: Admin (tab) -> Sites (expand item) -> Add New (item) -> Enter Name, Description, and select Preview Method.

Add User to a site: Site Admin (tab) -> Users (item) -> Username (textbox) [enter user from Grant SiteGod ACL section] -> select user (link) -> Edit (link) -> select roles (multi-select)



1.10 Import Four Packages using Catalog Mover

Add four packages using Catalog Mover. We will need to load some HTML files using Catalog Mover to trigger certain automated processes.

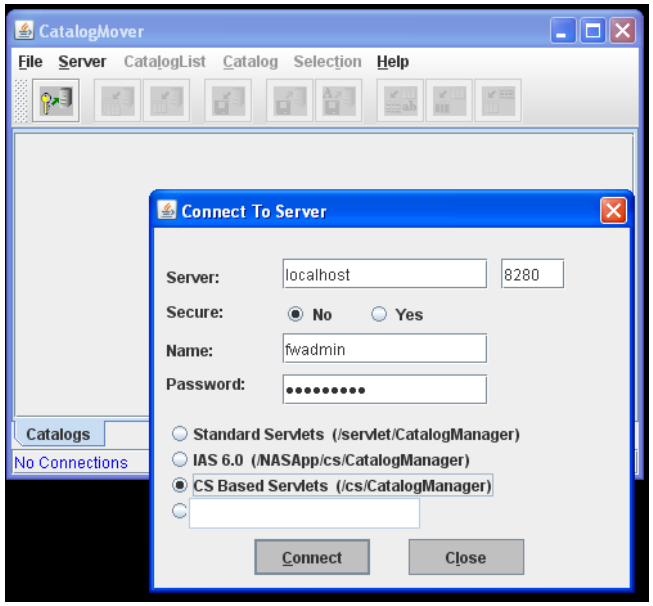
Catalog Mover can be located in your JSK's Content Server directory. Ultimately this section will create 2 entries in your CS Database.

Tables -> SiteCatalog -> GST -> Foundation -> entry for TagRegistryInstaller and entry for UrlRegistryInstaller

Example, on my machine it would be: `/Users/rgill/Library/JSK_CS753_CLEAN/ContentServer/7.5.3`

File: `catalogmover.sh` (`catalogmover.bat` for Windows)

Load Catalog Mover -> Server -> Connect -> Add your CS server and port credentials -> Secure No ->User credentials -> select CS Based Servlets -> Connect



Go to Catalog -> Import Catalog

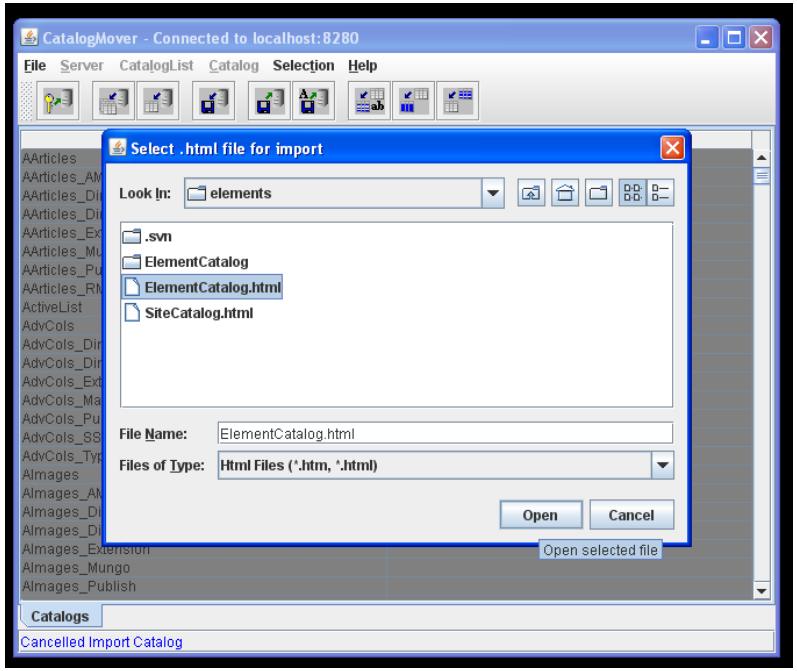
For the gsf-tagging module we need to import ElementCatalog.html and SiteCatalog.html

For the gsf-wra module we need to import ElementCatalog.html and SiteCatalog.html

These files will be located in

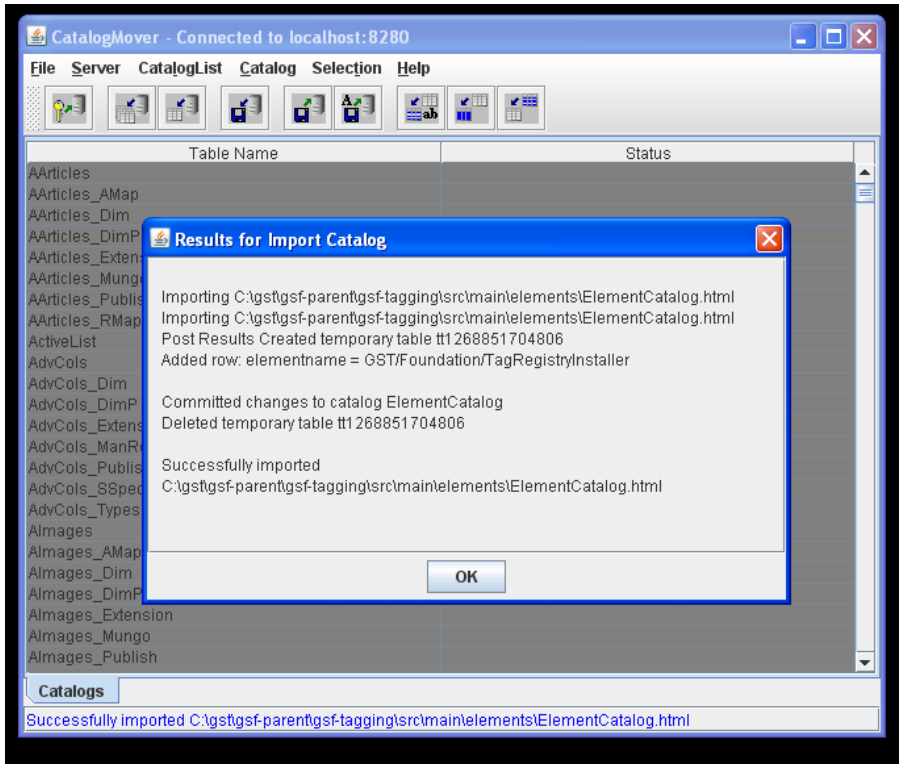
`gst-foundation/gsf-tagging/src/main/elements`
(ex: `/Users/rgill/Library/gsf/gst-foundation/gsf-tagging/src/main/elements`)

`gst-parent/gsf-wra/src/main/elements`
(ex: `/Users/rgill/Library/gsf/ gst-parent/gsf-wra/src/main/elements`)



Select your choice for Catalog Data Directory (New Catalogs Only) then press OK. It should be imported successfully.

Remember: this must be done for both gsf-tagging and gsf-wra and each must import 2 catalogs each.



1.11 Install GSF Event Listeners

We need to install the URL Registry and Tag Registry events. For example these events are triggered on a save of a Web-Referenceable Asset (WRA) asset.

In order to activate these events you need to trigger them through your browser. This requires SiteGod ACL.

Note: You can login with an account that has SiteGod ACL then set the URL to each of the Registry event JSP and trigger them.

The URL has the following structure: http://<server>:<port>/ cs/ContentServer?pagename=GST/Foundation/ TagRegistryInstaller

http://<server>:<port>/ cs/ContentServer?pagename=GST/Foundation/UrlRegistryInstaller

Example: http://localhost:8280/cs/ContentServer?pagename=GST/Foundation/UrlRegistryInstaller
http://localhost:8280/cs/ContentServer?pagename=GST/Foundation/TagRegistryInstaller

Note: When you execute the above 2 JSP, a blank white screen will appear, there will be no entries to your futuretense.txt log.

Check the screen and the log to confirm no errors occurred for both JSPs. Future versions will provide better feedback to you.

You can test to see if these Events have been registered by checking the AssetListener_reg table.

To do so have your ContentServer running and put this URL into your browser:

http://<server>:<port>/cs/CatalogManager?ftcmd=selectrow(s)&tablename=AssetListener_reg

example: http://localhost:8280/cs/CatalogManager?ftcmd=selectrow%28s%29&tablename=AssetListener_reg

id	listener	blocking
1153937286234	com.openmarket.basic.event.SearchAssetIdEventListener	Y
1285861083117	com.fatwire.gst.foundation.url.WraAssetEventListener	Y
1285861094618	com.fatwire.gst.foundation.tagging.TaggedAssetEventListener	Y
1285861094621	com.fatwire.gst.foundation.tagging.CacheMgrTaggedAssetEventListener	Y

Note: CacheMgrTaggedAssetEventListener is an asset event listener. On an asset save it will flush the cache accordingly. This works well for development environments. However, for a Delivery server this can be problematic. During a publish, if an asset is flushed with the CacheMgrTaggedAssetEventListener this can cause assets to be automatically flushed from cache while the publish is underway. Requests to pages may yield broken or incomplete content. To avoid this, delete the CacheMgrTaggedAssetEventListener. A RealTime Publishing Cache Updater publishing hook will be added that executes the cache clearing for the tagging service on the target of a publish. Once the publishing has been completed the Cache clearing will begin.

1.12 Configuring Realtime Publishing Cache Updater Plugin

This activity needs to be completed only on a delivery environment, or on an environment that is the target of a RealTime Publishing system.

Details coming soon.

1.13 Create GST Flex Family Asset

A GST Flex Family must be created with 2 particular types of child assets. This is mandatory for the GST to function in its entirety. For Description and Plural Form follow the convention you employ. These values are not critical for functionality.

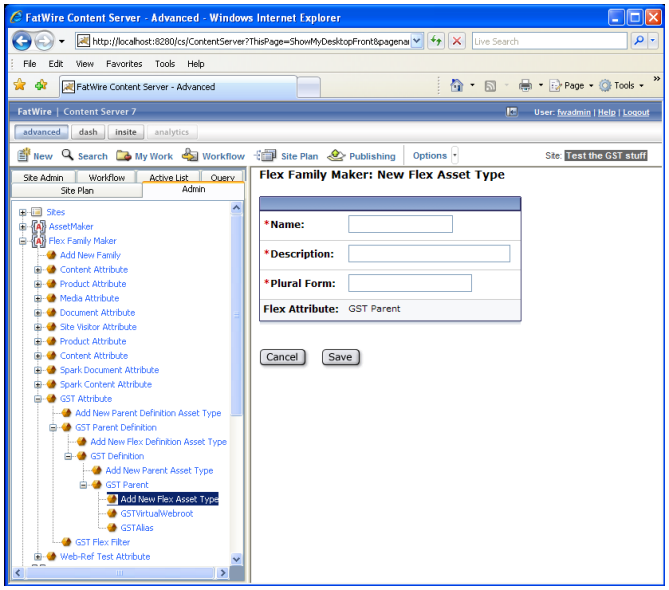
The structure for the Flex Family is the following:

Flex Attribute name: GSTAttribute
Flex Parent Definition name: GSTPDefinition
Flex Definition name: GSTDefinition
Flex Parent Asset name: GSTParent
Flex Asset name: GSTVirtualWebroot
Flex Asset name: GSTAlias
Flex Filter name: GSTFilter

Note: When you create a Flex Family it will only allow you to create one Flex Asset. It does not matter if you create GSTVirtualWebroot or GSTAlias first.

To create the 2nd Flex Asset go to: Admin (tab) -> expand GST Attribute > expand GST Parent Definition -> expand GST Definition -> expand GST Parent

-> Add New Flex Asset Type -> create the 2nd Flex Asset



We need to create GST Attributes which will be added to our 2 definitions (GSTVirualWebroot and GSTAlias)

Note: You specify if an attribute is required when you add it to a Definition not when you create the Attribute itself.

Create the following GST Attributes:

Name	**Value Type**	**Required?**
env_name	string	Yes
env_vwebroot	string	Yes
master_vwebroot	string	Yes
linktext	string	No
linkimage	string	No
popup	int (0/1)	Yes
target_url	string	No

Now create the following GST Definitions:

i) GSTVirtualWebroot

GST Parent Definitions: <empty>
Attributes: env_name, env_vwebroot, master_vwebroot
Filters: <empty>

ii) GSTAlias

GST Parent Definitions: <empty>
Attributes: target_url, popup, linktext, linkimage
Filters: <empty>
Single-valued Named Association of type “Any”: target

1.14 Configure Vanity URL Support

Vanity URLs are URLs that are defined entirely within the web referenceable asset. There is no restriction on the form of the URL as long as it is valid and the application server gets a chance to process it. By default, Content Server is configured so that only URLs starting with `/cs/` are forwarded to the Content Server application. For Vanity URLs, however, this is not sufficient. We must introduce smarter mapping to be able to do more.

Apache's mod_rewrite component is powerful enough, but Apache is not present by default on a Jump Start Kit. Instead, we have to use a servlet filter called Tuckey's URL Rewrite, which offers similar functionality. This document describes how to use both.

1.14.1 Enable the URL Assembler

Edit your `ServletRequest.properties` file located in your `JSK/AppServer/apache/webapps/cs/WEB-INF/classes`
For example: `/Users/rgill/Library/JSK_CS753_CLEAN/App_Server/apache-tomcat-6.0.18/webapps/cs/WEB-INF/classes`

Set the following properties:

```
com.fatwire.gst.foundation.url.wrapathassembler.dispatcher=GST/Dispatcher
uri.assembler.1.shortform=wrapath
uri.assembler.1.classname=com.fatwire.gst.foundation.url.WraPathAssembler
uri.assembler.2.shortform=query
uri.assembler.2.classname=com.fatwire.cs.core.uri.QueryAssembler
```

Note: You should comment out the original lines that you are replacing.

1.14.2 Install and Configure Tuckey Servlet Url Rewrite

Note that if you are using Apache, please see "Apache mod_rewrite Examples" instead.

Before we begin this process there are 2 things to consider.

- Determine whether you would like to "**forward**" or "**redirect**" requests by the Application Server.
- Determine the the context path for both Tuckey and Content Server

1.14.2.1 Forward vs Redirect

forward: performed internally by servlet, browser is unaware, repeats request with the same URL (pretty URL retained), potential for some efficiency gains. Tuckey **must** be installed in the **same** web application as Satellite Server for this configuration to be possible.

redirect: 2 step process where Application Server tells the browser to obtain a second URL (original URL) instantaneously. The vanity URL is lost, and each URL results in a second request, so it is less efficient. Tuckey **can** be installed in **any** web application for this configuration to be possible.

Forward Example on a JSK
User enters the URL, http://localhost:8280/cs/products/make/model123.html
The browser would show after the request has finished, http://localhost:8280/cs/products/make/model123.html
Bookmarks and search websites will see the above URL.

Redirect Example on a JSK
User enters the URL, http://localhost:8280/products/make/model123.html
The browser would show after the requests have finished:
http://localhost:8280/cs/Satellite?pagename=GST/Dispatcher&virtual-webroot=http://localhost:8280/cs&url-path= products/make/model123.html
Bookmarks and search websites will see the above URL.

1.14.3 Context Path Determination

There are two parts to this: Tuckey & Content Server. As described above, there are advantages to having Tuckey deployed in the same webapp as Content Server. The drawback is that all URLs must be prefixed with something that will allow them to be accessed by both Tuckey and CS. On a JSK or a standard Content Server installation, this is `/cs/`. Moving the app is possible, but complicated, and is therefore not covered by this document.

Option 1: **`cs`**

By default Content Server installs with the '`cs`' Context Path.

Thus, the path may be something like: `/Users/rgill/Library/JSK_CS753_CLEAN/App_Server/apache-tomcat-6.0.18/webapps/cs`

The consequence of this is the environment virtual webroot must contain "`/cs`" in its path.

You might have your virtual webroot asset similar to this:

Environment Name: fatwire-dev
Environment Web Root: http://localhost:8280/cs/products
Master Virtual Web Root: http://your.domain.name/products

WRA's path: http://your.domain.name/products/make/model123.html
Example URL: http://localhost:8280/cs/products/make/model123.html

Note: The resultant browser URL will depend on how the Tuckey URLRewrite is configured (forward or redirect). If you put Tuckey into the `ROOT` web application context path, you can set your environment virtual webroot however you want (importantly, to a value identical to the master virtual webroot).

Option 2: **`ROOT`**

By having Content Server installed in here with Tuckey URLRewrite, you can get pretty URLs to be the same as above except without "`/cs/`" since the Context Path is `ROOT`. As a result the environment virtual webroot would not need to have the "`/cs`" included in their path attribute.

Example location of ROOT would be: `/Users/rgill/Library/JSK_CS753_CLEAN/App_Server/apache-tomcat-6.0.18/webapps/ROOT`

You might have you virtual webroot asset similar to this:

Environment Name: fatwire-dev
Environment Web Root: http://localhost:8280/products
Master Virtual Web Root: http://your.domain.name/products

WRA's path: http://your.domain.name/products/make/model123.html
Example URL in browser: http://localhost:8280/products/make/model123.html (with Tuckey in forward mode)

This deployment is more complicated, and is not necessary when Apache is installed.

Note: Installing Apache and configuring mod_rewrite has the benefit of deploying Tuckey in the ROOT context, and the performance and user experience of configuring Tuckey in Forward mode, and is therefore the preferred configuration option.

Tuckey may be used with Remote Satellite Server, however, by deploying both in the ROOT context of the Remote Satellite Server system.

1.14.4 Tuckey Installation Details

Regardless if you choose to install your Content Server in the "cs" or "ROOT" the exact same process must take place relative to each. When configuring and installing Tuckey you must edit the `web.xml` of your Context Path's located in its `WEB-INF`. You must place the `urlrewrite.xml` also in the same directory as the `web.xml` (unless you wish to do additional configuration). You must put the Tuckey JAR in the `WEB-INF/lib` directory.

For this example we will install and configure Tuckey in the 'cs' Context Path.

For example: `/Users/rgill/Library/JSK_CS753_CLEAN/App_Server/apache-tomcat-6.0.18/webapps/cs/WEB-INF`

Download the Tuckey UrlRewrite Servlet Filter version 3.2.

Please go to: http://www.tuckey.org/urlrewrite/

Manual for version 3.2 can be found at: http://urlrewritefilter.googlecode.com/svn/trunk/src/doc/manual/3.2/index.html

Follow these steps for configuration. Download, unzip and put the jar in your Application Server's Context's `WEB-INF lib` directory.

`ex: /Users/rgill/Library/JSK_CS753_CLEAN/App_Server/apache-tomcat-6.0.18/webapps/cs/WEB-INF/lib/urlrewrite-3.2.0.jar`

Next edit your `web.xml` and add the filter settings outlined in the site. Also ensure that the urlrewrite.xml for Tuckey is located in the same directory as your `web.xml` (unless you wish to do additional configuration). You can find your web.xml in your Application Server's Context WEB-INF directory.

`ex: /Users/rgill/Library/JSK_CS753_CLEAN/App_Server/apache-tomcat-6.0.18/webapps/cs/WEB-INF`

Note: It is suggested to have the Tuckey Filter as the first <filter> tag in your web.xml

Example Tuckey Filter configuration:

```
<filter>
      <filter-name>UrlRewriteFilter</filter-name>
      <filter-class>org.tuckey.web.filters.urlrewrite.UrlRewriteFilter</filter-class>
      <!-- If you wish to enable logging set the level -->
      <init-param>
            <param-name>logLevel</param-name>
            <param-value>INFO</param-value>
      </init-param>
      <!-- set the amount of seconds the conf file will be checked for reload
          can be a valid integer (0 denotes check every time,
                empty/not set denotes no reload check) -->
      <init-param>
            <param-name>confReloadCheckInterval</param-name>
            <param-value>0</param-value>
      </init-param>
</filter>
<filter-mapping>
      <filter-name>UrlRewriteFilter</filter-name>
      <url-pattern>/*</url-pattern>
      <dispatcher>REQUEST</dispatcher>
      <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

The final step is to add the URLRewrite rules. Each rule corresponds directly to the value defined in the environment virtual webroot part of your Virtual Webroot assets. Therefore this section is defined in Part II.

2 PART II: USING THE GST SITE FOUNDATION

2.1 Define Your Web-Referenceable Asset

Both Basic and Flex Asset can be WRA. Description and Plural Form follow the convention you employ. These values are not critical for functionality.

Note: You MUST provide a template value each time you create a Web-Referenceable Asset. This Installation Guide uses WRA Flex Assets only. However, for reference, an example Basic Asset Asset Descriptor File is provided:

```
<?xml version="1.0" ?>
<ASSET NAME="WebRefTest" DESCRIPTION="Web-Referenceable Basic Asset">
<PROPERTIES>
	<PROPERTY NAME="gsttag" DESCRIPTION="gsttag">
		<STORAGE TYPE="VARCHAR" LENGTH="255" />
		<INPUTFORM TYPE="TEXT" DESCRIPTION="gsttag" REQUIRED="NO" />
		<SEARCHFORM TYPE="TEXT" DESCRIPTION="gsttag" />
		<SEARCHRESULTS INCLUDE="TRUE" />
	</PROPERTY>
	<PROPERTY NAME="metatitle" DESCRIPTION="metatitle">
		<STORAGE TYPE="VARCHAR" LENGTH="255" />
		<INPUTFORM TYPE="TEXT" DESCRIPTION="metatitle" REQUIRED="YES" />
		<SEARCHFORM TYPE="TEXT" DESCRIPTION="metatitle" />
		<SEARCHRESULTS INCLUDE="TRUE" />
	</PROPERTY>
	<PROPERTY NAME="metadescription" DESCRIPTION="metadescription">
		<STORAGE TYPE="VARCHAR" LENGTH="255" />
		<INPUTFORM TYPE="TEXT" DESCRIPTION="metadescription" REQUIRED="YES" />
		<SEARCHFORM TYPE="TEXT" DESCRIPTION="metadescription" />
		<SEARCHRESULTS INCLUDE="TRUE" />
	</PROPERTY>
	<PROPERTY NAME="metakeyword" DESCRIPTION="metakeyword">
		<STORAGE TYPE="VARCHAR" LENGTH="255" />
		<INPUTFORM TYPE="TEXT" DESCRIPTION="metakeyword" REQUIRED="NO" />
		<SEARCHFORM TYPE="TEXT" DESCRIPTION="metakeyword" />
		<SEARCHRESULTS INCLUDE="TRUE" />
	</PROPERTY>
	<PROPERTY NAME="h1title" DESCRIPTION="h1title">
		<STORAGE TYPE="VARCHAR" LENGTH="255" />
		<INPUTFORM TYPE="TEXT" DESCRIPTION="h1title" REQUIRED="YES" />
		<SEARCHFORM TYPE="TEXT" DESCRIPTION="h1title" />
		<SEARCHRESULTS INCLUDE="TRUE" />
	</PROPERTY>
	<PROPERTY NAME="linktext" DESCRIPTION="linktext">
		<STORAGE TYPE="VARCHAR" LENGTH="255" />
		<INPUTFORM TYPE="TEXT" DESCRIPTION="linktext" REQUIRED="NO" />
		<SEARCHFORM TYPE="TEXT" DESCRIPTION="linktext" />
		<SEARCHRESULTS INCLUDE="TRUE" />
	</PROPERTY>
</PROPERTIES>
</ASSET>
```

Example for a Flex Family you can create a new Flex Family Asset.

Flex Attribute name: WebRefTestA
Flex Parent Definition name: WebRefTestPD
Flex Definition name: WebRefTestD
Flex Parent name: WebRefTestP
Flex Asset name: WebRefTest
Flex Filter name: WebRefTestF

Create the following new attributes, these are required to make this asset a Web-Referenceable.

WebRefTestA (new Attributes for the Flex Family WebRefTest)

Note: You specify if an attribute is required when you add it to a Definition not when you create the Attribute itself.

Name	**Value Type**	**Required?**
Gsttag	string	No
h1title	string	Yes
linktext	string	No
metadescription	string	Yes
metakeyword	string	No
metatitle	string	Yes

Create a child definition for WebRefTestCD. It must include the above fields.

Example: Name: WebRefTestAssetDef
Parent Definitions: <empty>
Attributes: all of the above.
Filters: <empty>

Click Save Changes button. This creates a Web-Referenceable Asset to be utilized by GST.

2.2 Create GST Dispatcher (Wrapper Page)

The GST/Dispatcher CSElement is used as a Wrapper.

Step 1. Create a new CSElement (xml). Note: do not add extra white-spaces to the xml.

*Name: GSTDispatcher
Description: GST Dispatcher (Wrapper Page)
*Rootelement: GSTDispatcher
*Element Storage Path/Filename: GSTDispatcher.xml
*Element Logic (replace any existing code with this):

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd"><FTCS Version="1.1"><CALLJAVA
CLASS="com.fatwire.gst.foundation.controller.BaseController"/></FTCS>
```

Step 2. First add the above CSElement to your ActiveList.

Create a new Site Entry Asset (Enable Site Entry Asset Type if it has not already been enabled)

*Name: GST/Dispatcher
Description: GST Dispatcher
Add the GSTDispatcher CSElement (Add Selected Items).
Adding the CSElement will give the following, *Rootelement: GSTDispatcher
*Pagename: GST/Dispatcher
Wrapper Page: Yes
Cache Rule: Uncached

Now there is a site entry for GST/Dispatcher.

2.3 Working With Virtual Webroots and Vanity URLs

2.3.1 Overview

Virtual Webroots are used to control how vanity URLs are interpreted by your system. A virtual webroot is the part of a URL that represents a namespace on your site that will always be serviced by Content Server.

Each environment (dev, staging, production, etc) has its own virtual webroot. The virtual webroot for the production-delivery server usually also corresponds to what is referred to as the "master virtual webroot".

When a URL is defined in a WRA, the namespace for that URL is entered directly into the "path" field of the WRA. The asset-specific part immediately follows. For example for a product "fPad", one might use a url such as: http://your.domain.name/products/fPad. In this case, the master virtual webroot would be http://your.domain.name/products since all products can map under this location.

How URLs are created (assembled)

On the editorial environment, the environment-specific virtual webroot might look quite a bit different: http://staging.domain.name:7001/cs/products. Because the asset is defined with the master, and because each environment is defined through a system property, the vanity URL system is able to rewrite the URL recorded in the "path" field and replace the master virtual webroot with the environment-specific virtual webroot for the appropriate environment when the URL is created.

This is how vanity URLs are created and how they can work across multiple environments.

To summarize:

- each server must have its environment name configured
- assets' URLs are defined such that the first part of the URL corresponds to a master virtual webroot
- an environment-specific value for the virtual webroot must be defined for each environment so that the GSF can substitute the environment-specific part of the URL in place of the master.

This means that there will be one virtual webroot asset for each virtual webroot for each environment. If there are 5 virtual webroots and 3 environments, there must be 15 instances of virtual webroot assets.

How URLs are resolved (disassembled)

On the URL disassembly side, things are a bit different. As described in the installation section above, requests must match a particular namespace in order to be able to reach Content Server – that's how the application server is handed the request. However the namespace that the application server is expecting may not (will probably never) match what the marketing team would like to see in the URLs.

The URLs must be re-written before they are screened for forwarding to the Content Server application.

This is where the idea of virtual webroots becomes very handy. On any given environment, the env_vwebroot (environment-specific virtual webroot) can be used to define a rewrite rule. Each virtual webroot can be used to convert the beautiful (yet app-server-inaccessible) vanity URL into a URL that the application server can handle.

In fact, we have enough information inside Content Server to automatically generate those rules for you! (unfortunately, that tool hasn't been written yet, but it's "on the short list"!)

For each env_vwebroot, a mod_rewrite rule is generated that splits the incoming URL into two parts – the virtual-webroot and the url-path. These two parts are then re-positioned as query string parameters, and the base URL is hard-coded to be a normal (non-vanity) url to the Satellite Server servlet on Content Server, with the pagename set to the dispatcher that's defined in the URL assembler (pagename=GST/Dispatcher is the default).

2.3.2 Create a Virtual Webroot

We need to now create a GSTVirtualWebroot asset instance:

Name: Products Virtual Webroot
master_vwebroot: Enter your master virtual web root, example: http://your.domain.name/products
env_vwebroot: Enter your environment web root, example: http://<server name>:<port>, example: http://localhost:8280/cs/products
env_name: Enter the environment name, set in **Pretty URL and Tuckey UrlRewrite Setup** section , example: fatwire-dev

Note: The environment name was defined in the chapter entitled "Install and Configure Content Server". Please be sure to use the same name.

2.3.3 Create the Rewrite Rules for the Virtual Webroot

Now that the virtual webroot is defined, we need to configure the web server (or Tuckey) to forward those requests to Content Server (or redirect them if Tuckey is configured in a different webapp than Content Server).

You need to define a regular expression that parses the incoming URL and can split it into virtual-webroot and url-path. Virtual-webroot is defined in Content Server as the env_vwebroot field in the Virtual Webroot (http://localhost:8280/cs/products in the example above). This value should be hard-coded into the rewrite rule. The base URL should be the normal path to access the dispatcher page through Satellite Server (http://localhost:8280/cs/Satellite?pagename=GST/Dispatcher is a common example).

Note: Whenever the env_vwebroot field in a Virtual Webroot asset is modified, the rewrite rules need to be updated on all affected environments.

2.3.3.1 Tuckey rewrite examples

Forward

```
<rule match-type="wildcard">
       <from>http://localhost:8280/cs/products/**</from>
       <to type="forward">http://localhost:8280/cs/Satellite?pagename=GST/Dispatcher&amp;virtual-webroot=http://localhost:8280/cs/products&amp;url-
path=$1</to>
</rule>
```

Redirect

```
<rule match-type="wildcard">
       <from>http://localhost:8280/cs/products/**</from>
       <to type="redirect">http://localhost:8280/cs/Satellite?pagename=GST/Dispatcher&amp;virtual-webroot=http://localhost:8280/cs/products&amp;url-
path=$1</to>
</rule>
```

2.3.3.2 Apache mod_rewrite Examples

Coming soon.

Comment [DD1]: It is pretty important to have this here soon.

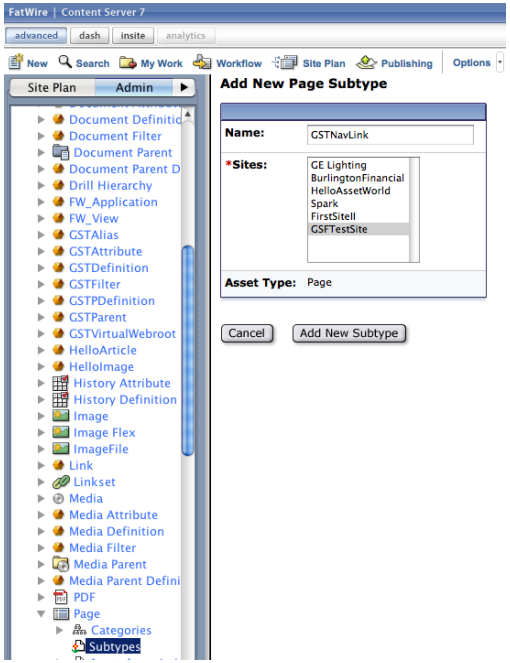
2.4 Building Site Navigation and Maps

With the GST there is a simple way to produce navigation utilizing Web-Referenceable Assets.

I will outline an example that uses the GST Navigation features.

Step 1. Create a new Page sub-type called “GSTNavName”. This name is mandatory.
To create a new Page type: Admin → Asset Types → Page → Subtypes → Add New Subtype → Name: GSTNavName → [select GSTTestSite] → click Add New Subtype

Step 2. Create another Page sub-type called “GSTNavLink”. This name is mandatory.
To create a new Page Type: Admin → Asset Types → Page → Subtypes → Add New Subtype → Name: GSTNavLink → [select GSTTestSite] → click Add New Subtype



Step 3. Create a test page navigation Template.

Sample Name: GSTPageNavigation
Create Element Type: JSP
Asset Type: Any [Typeless]

Add the following code in the appropriate Element Logic section:

```
<%@ page import="java.util.ArrayList,
                 java.util.Map,
                 COM.FutureTense.Interfaces.*,
                 com.fatwire.gst.foundation.taglib.NavigationHelper"
%><%
    AssetData ad = AssetDataUtils.getAssetData(ics.GetVar("c"), ics.GetVar("cid"),
            "metatitle", "metadescription", "metakeyword", "h1title", "linktext");
%>
<html>
<head>
    <meta name="title" content='<%=AttributeDataUtils.asString(ad.getAttributeData("metatitle"))%>' />
    <meta name="description" content='<%=AttributeDataUtils.asString(ad.getAttributeData("metadescription"))%>' />
    <meta name="keyword" content='<%=AttributeDataUtils.asString(ad.getAttributeData("metakeyword"))%>' />
<title><%=AttributeDataUtils.asString(ad.getAttributeData("metatitle"))%> | <ics:getvar name="site" /></title>
</head>
<%-- Example only goes 1 level deep under GSTNavName --%>
<body>

       <div id="nav">
          <asset:load name="main-nph" type="Page" field="name" value="GSTNavName"/>
          <asset:get name="main-nph" field="id" output="main-nph-id"/>
          <%-- No GSTNavName or No GSTNavLink under GSTNavName produces a NullPointerException --%>
          <%
              NavigationHelper nh = new NavigationHelper(ics);
              Map<String, Object> nav = nh.getSitePlanAsMap(ics.GetVar("main-nph-id"));
          %>
          <ul>
          <%
              for(Map<String,Object> kid : (ArrayList<Map<String,Object>>)nav.get("children")) {
          %>
              <li>
                  <a href='<%=kid.get("url")%>'>
                      <%=kid.get("linktext")%>
                  </a>
              // nested menus can be added here
              </li>
          <%
              }
          %>
       </div>
</body>
</html>
```

Note: If the linktext attribute is empty it will then display the h1title attribute. This is why the screenshot example upcoming shows text from different attributes. WRA #1 displays its h1title and WRA #2 displays it linktext.

Step 4. Create a new Web-Referenceable Asset (WRA) that has its Template attribute set to the above Template created. This is the asset we will preview later on to view the Navigation.

New → New Web Referencable Test Asset → Name: WRANavTest
→ Description: This WRA is to test the Navigation
→ Select A Template: /GSTPageNavigation
→ h1title: Navigation Test
→ metadescription: Testing the Navigation Feature
→ metatitle: TEST!

Step 5. Create a GSTNavName Page Asset.

New → New Page → Name: GSTNavName
→ Description: GST Navigation
→ Subtype: GSTNavName

Step 6. Create a GSTNavLink Page Asset for WRA #1 and WRA #2.

First lets add WRA #1 and WRA #2 to "My Work" – "My Active List" list of assets.

Search → Find Web Referenceable Test Asset → Search → [far right column check the checkbox beside WRA #1 & WRA #2] → Add to My Active List

Now we will create our GSTNavLink for WRA #1 and WRA #2.

New → New Page → Name: NavWRA1

→ Description: Navigation Link for WRA #1

[In the left toolbar menu choose Active List]

→ select WebRefTestAsset1 on your Active List and under Contains click Add Selected Items. Save the asset.

New → New Page → Name: NavWRA2

→ Description: Navigation Link for WRA #2

[In the left toolbar menu choose Active List]

→ select WebRefTestAsset2 on your Active List and under Contains click Add Selected Items. Save the asset.

Step 7. Place our pages to the site.

In the left toolbar menu choose Site Plan.

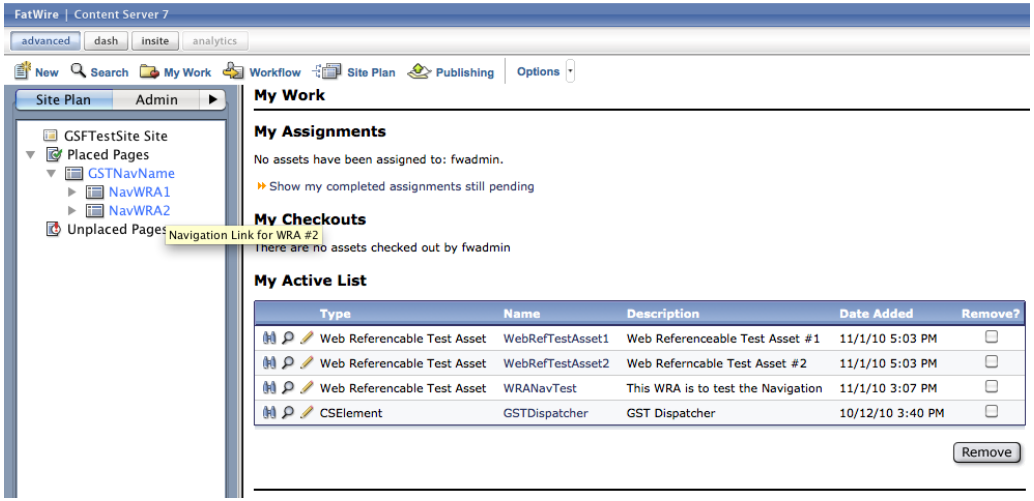
(You may have to Right Click and Refresh / Refresh All to reload assets created).

Right Click on Place Pages → For GSTNavName give it a rank 0 → click Save

Right Click and click Refresh / Refresh All

Placed Pages → GSTNavName should appear → Right Click GSTNavName → Place Page → For NavWRA1 give it a rank 0 →

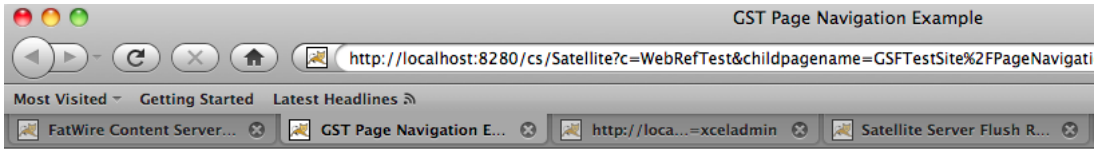
For NavWRA2 give it a rank 1 → click Save



You can now Refresh / Refresh All and will have to following hierarchy: Place Pages → GSTNavName → NavWRA1, NavWRA2

Step 8. Preview the WRANavTest Web-Referenceable Asset.

You should get a screen like this:



Note: The links use the nice URLs. Notice the URL at the bottom of the browser that is from an on hover on the second link.

Note: "WebRefAsset1-H1Title" is the h1title attribute from WRA #1 since this asset has no linktext value. "someLinkText" is the linktext value from WRA #2.

2.5 Using JSP Tablib

In order to use the GST Tablib you need to include the tablib in your JSP. This is done by including the following line in your JSP:

```
<%@taglib uri="http://gst.fatwire.com/foundation/tags/gsf" prefix="gsf"%>
```

Where prefix can be a user defined value, but standard is to use gsf. By including this tablib you have access to the GST tags which are `<gsf:asset-tagged-list>` and `<gsf:tagged-list>`.

Note: you must have successfully completed the section Add GST jar to Content Server in order to use the GST Taglib.

gsf-tagged-list:

```
<gsf:tagged-list tag=”someValue” outlist=”nameOfOutputList” />
```

This tag searches the GSTTagRegistry table on the attribute ‘tag’ for values matching “someValue” and returns the results as an IList in “nameOfOutputList”. GSTTagRegistry results returned must valid date range.

gsf-asset-tagged-list:

```
<gsf:asset-tagged-list assetid=”someId” assettype=”someType” outlist=”someOutputList” />
```

This tag searches the GSTTagRegistry table on the attribute ‘tag’ for values matching the following pattern: asset-someId:someType.

Note: It can be any asset type not just Web-Referenceable Assets (WRA) stored in the gsttag attribute.

For example: tag = asset-126885192013:WebRefTest

When called: `<gsf:asset-tagged-list assetid=”126885192013” assettype=”WebRefTest” outlist=”webRefList” />`

A Note about Date Ranges:

A valid date range is the following:

startDate = null & endDate = null

startDate = null, effectiveDate is before endDate (effectiveDate = null will be interpreted as effectiveDate = current time)

endDate = null, startDate is before effectiveDate (effectiveDate = null will be interpreted as effectiveDate= current time)

startdate is before effectiveDate and effectiveDate is before endDate (effectiveDate = null will be interpreted as effectiveDate= current time)

3 PART III: TESTING

3.1 Test WRA Support

Note: This test uses the Flex Asset WRA.

Create a new WRA (WebRefTest for example).

Name: WebRefTestAsset1
Description: Web Referenceable Test Asset #1
Select a Template: <will set once we create the Web-Referenceable Test Template example, so EMPTY for now>
Path: http://localhost:8280/cs/niceURL (the webroot must match the master virtual webroot in your GSTVirtualWebroot asset)
Start Date: 2010-01-01 15:31:18
End Date: 2020-01-01 15:31:18
*h1title: WebRefAsset1-H1Title
linktext:
*metadescription: WebRefAsset1-metadescription
*metatitle: WebRefAsset1-metatitle
gsttag: gstTag1,gstTag2,tagValue1

Create another new WRA (WebRefTest for example).

Name: WebRefTestAsset2
Description: Web Referenceable Test Asset #2
Select a Template: <will set once we create the JSP Taglib Template example, so EMPTY for now>
Path: <empty is fine>
Start Date: 2010-01-01 15:31:18
End Date: 2020-01-01 15:31:18
*h1title: WebRefAsset2-H1Title
linktext: someLinkText
*metadescription: WebRefAsset2-metadescription
*metatitle: WebRefAsset2-metatitle
gsttag: tagValue1,tagValue2, asset-<assetId from WRA #1>:WebRefTest

On save of WRA #1 will be a new entry in the GSTUrlRegistry since there exists a value for the path attribute.

opt_vwebroot must match a valid master_vwebroot of a GSTVirtualWebroot asset.

If you do a table dump of GSTUrlRegistry you should get something like:

site: http://<server>:<port>/cs/CatalogManager?ftcmd=selectrow%28s%29&tablename=GSTUrlRegistry

ex: http://localhost:8280/cs/CatalogManager?ftcmd=selectrow%28s%29&tablename=GSTUrlRegistry

id	Path	assettype	assetid	Startdate	enddate	opt_vwebroot	opt_url_path	opt_depth	opt_site
1287167995232	http://localhost:8280/cs/ContentServer/niceURL	WebRefTest	1268851692013	2010-10-08 15:31:18	2010-10-08 15:31:18	http://localhost:8280	/cs/ContentServer/niceURL	4	GSFTestSite

Also on save it will create 3 entries in the GSTTagRegistry for WRA #1 since the gsttag attribute has 3 values, gstTag1, gstTag2,tagValue1.

It will create 3 entries in the GSTTagRegistry for WRA #2 since the gsttag attribute has 3 values: tagValue1,tagValue2,asset-1268851692013:WebRefTest

If you do a table dump of GSTTagRegistry you should get something like:

site: http://<server>:<port>/cs/CatalogManager?ftcmd=selectrow%28s%29&tablename=GSTTagRegistry

ex: http://localhost:8280/cs/CatalogManager?ftcmd=selectrow%28s%29&tablename=GSTTagRegistry

id	tag	assettype	assetid	startdate	enddate
1268851735918	gstTag1	WebRefTest	1268851692013	08/10/2010 15:31	09/10/2011 15:31
1268851735919	gstTag2	WebRefTest	1268851692013	08/10/2010 15:31	09/10/2011 15:31
1268851735920	tagValue1	WebRefTest	1268851692013	08/10/2010 15:31	09/10/2011 15:31
1268851739389	tagValue1	WebRefTest	1268851692778	01/10/2010 10:32	31/10/2011 10:32
1268851739390	tagValue2	WebRefTest	1268851692778	01/10/2010 10:32	31/10/2011 10:32
1268851739391	asset-1268851692013:WebRefTest	WebRefTest	1268851692778	01/10/2010 10:32	31/10/2011 10:32

Test Template

This test template displays the H1 Title attribute and makes a link to the very same asset. It tests that you are able to retrieve data from the Web-Referenceable Asset and that links (proper URL) can be created by the GST.

Create a new Template named: WebRefTemplateGeneric

For Asset Type: WebRefTest

Add to the appropriate area of Element Logic:

```
<%@ page import="com.fatwire.assetapi.data.AssetData,
                 com.fatwire.gst.foundation.facade.assetapi.AssetDataUtils,
                 com.fatwire.gst.foundation.facade.assetapi.AttributeDataUtils"
%>

<%
    AssetData ad = AssetDataUtils.getAssetData(ics.GetVar("c"), ics.GetVar("cid"),
            "metatitle", "metadescription", "metakeyword", "h1title", "linktext");
%>

<html>
<head>
    <meta name="title" content='<%=AttributeDataUtils.asString(ad.getAttributeData("metatitle"))%>' />
    <meta name="description" content='<%=AttributeDataUtils.asString(ad.getAttributeData("metadescription"))%>' />
    <meta name="keyword" content='<%=AttributeDataUtils.asString(ad.getAttributeData("metakeyword"))%>' />
<title><%=AttributeDataUtils.asString(ad.getAttributeData("metatitle"))%> | <ics:getvar name="site" /></title>
</head>

<body>
       <h1><%=AttributeDataUtils.asString(ad.getAttributeData("h1title"))%></h1>
       <render:gettemplateurl site='<%=ics.GetVar("site")%>' tname="WebRefTemplateGeneric" tid='<%=ics.GetVar("tid")%>' slotname="WebRefLink" c='<%=
ics.GetVar("c") %>' cid='<%= ics.GetVar("cid") %>' outstr="linkURL" />
       <a href='<%=ics.GetVar("linkURL")%>'> Link to same asset </a>
</body>
</html>
```

To use the JSP Tablib you can create a Template with the following code:

Create a new Template named: WebRefTaglibExample

For Asset Type: WebRefTest

Add to the appropriate area of Element Logic:

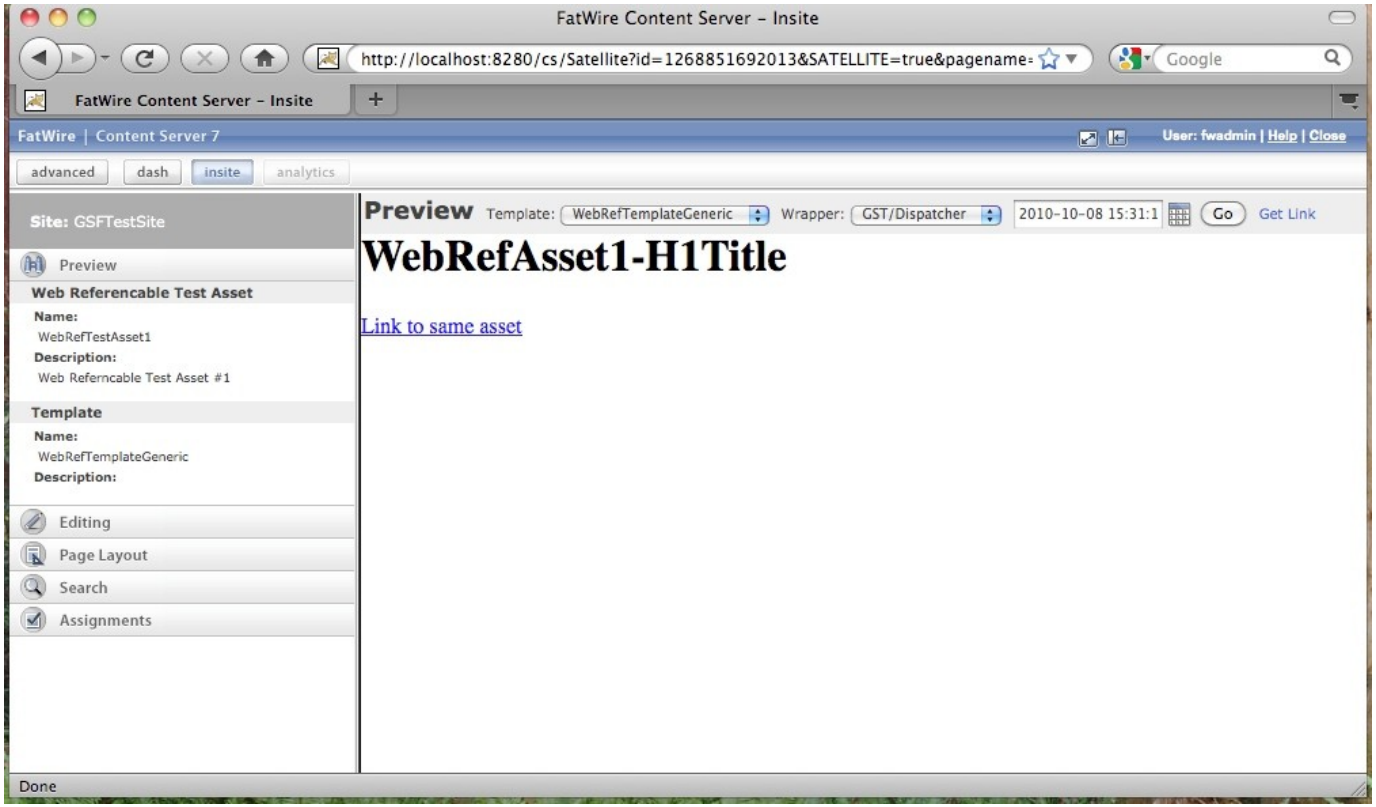
```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="gsf" uri=http://gst.fatwire.com/foundation/tags/gsf
%><%@ page import="COM.FutureTense.Interfaces.*,
                   com.fatwire.assetapi.data.AssetData,
                   com.fatwire.gst.foundation.facade.assetapi.AssetDataUtils,
                   com.fatwire.gst.foundation.facade.assetapi.AttributeDataUtils,
                   com.fatwire.gst.foundation.facade.assetapi.AssetIdIList,
                   com.fatwire.gst.foundation.facade.sql.*"
%><cs:ftcs>
<ics:if condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logdep cid='<%=ics.GetVar("tid")%>'
c="Template"/></ics:then></ics:if>
<%
    AssetData ad = AssetDataUtils.getAssetData(ics.GetVar("c"), ics.GetVar("cid"),
            "metatitle", "metadescription", "metakeyword", "h1title", "linktext");
%>
<html>
<head>
    <meta name="title" content='<%=AttributeDataUtils.asString(ad.getAttributeData("metatitle"))%>' />
    <meta name="description" content='<%=AttributeDataUtils.asString(ad.getAttributeData("metadescription"))%>' />
    <meta name="keyword" content='<%=AttributeDataUtils.asString(ad.getAttributeData("metakeyword"))%>' />
    <title><%=AttributeDataUtils.asString(ad.getAttributeData("metatitle"))%> | <ics:getvar name="site" /></title>
</head>
<body>
<%-- for the tag='', put in a value that exists in the tag column of the GSTTagRegistry table --%>
<%
    String tagColumnValue = "large";
%>
<%-- for valid row(s) in the GSTTagRegistry that has the tag attribute equal to tagColumnValue display the total count,
assetId and assetType --%>
<gsf:tagged-list tag="<%= tagColumnValue %>" outlist="gsfTaggedList" />
<%
```

```
    out.println("<br/><br/> Entries in gsf:tagged-list matching tag='"+tagColumnValue+"'");
      for (Row listRow : new IListIterable(ics.GetList("gsfTaggedList"))) {
            Long assetId = listRow.getLong(AssetIdIList.ASSETID);
            String assetType = listRow.getString(AssetIdIList.ASSETTYPE);
            out.println("<br/> assetId= "+assetId+", assetType="+assetType);
      }
%>


<%-- Find WRA1 since WRA2 gsttag attribute contains it (we should probably use asset:list instead)  --%>
<asset:load name="wraLoad" type="TestWRA" field="name" value="taxi" site='<%= ics.GetVar("site") %>'/>
<asset:get name="wraLoad" field="id" output="wraLoadId" />

<%-- Given a particular Asset Id and Asset Type, find valid GSTTagRegistry entries that contain this pairing
information --%>
<gsf:asset-tagged-list assetid='<%= ics.GetVar("wraLoadId") %>' assettype='TestWRA' outlist="gsfAssetTaggedList" />
<%
    out.println("<br/><br/> Entries in gsf:asset-tagged-list matching tag='asset
"+ics.GetVar("wraLoadId")+":TestWRA"+"'");
    for (Row listRow : new IListIterable(ics.GetList("gsfAssetTaggedList"))) {
          Long assetId = listRow.getLong(AssetIdIList.ASSETID);
          String assetType = listRow.getString(AssetIdIList.ASSETTYPE);
          out.println("<br/> assetId= "+assetId+", assetType="+assetType);
    }
%>
</body>
</html>
</cs:ftcs>
```

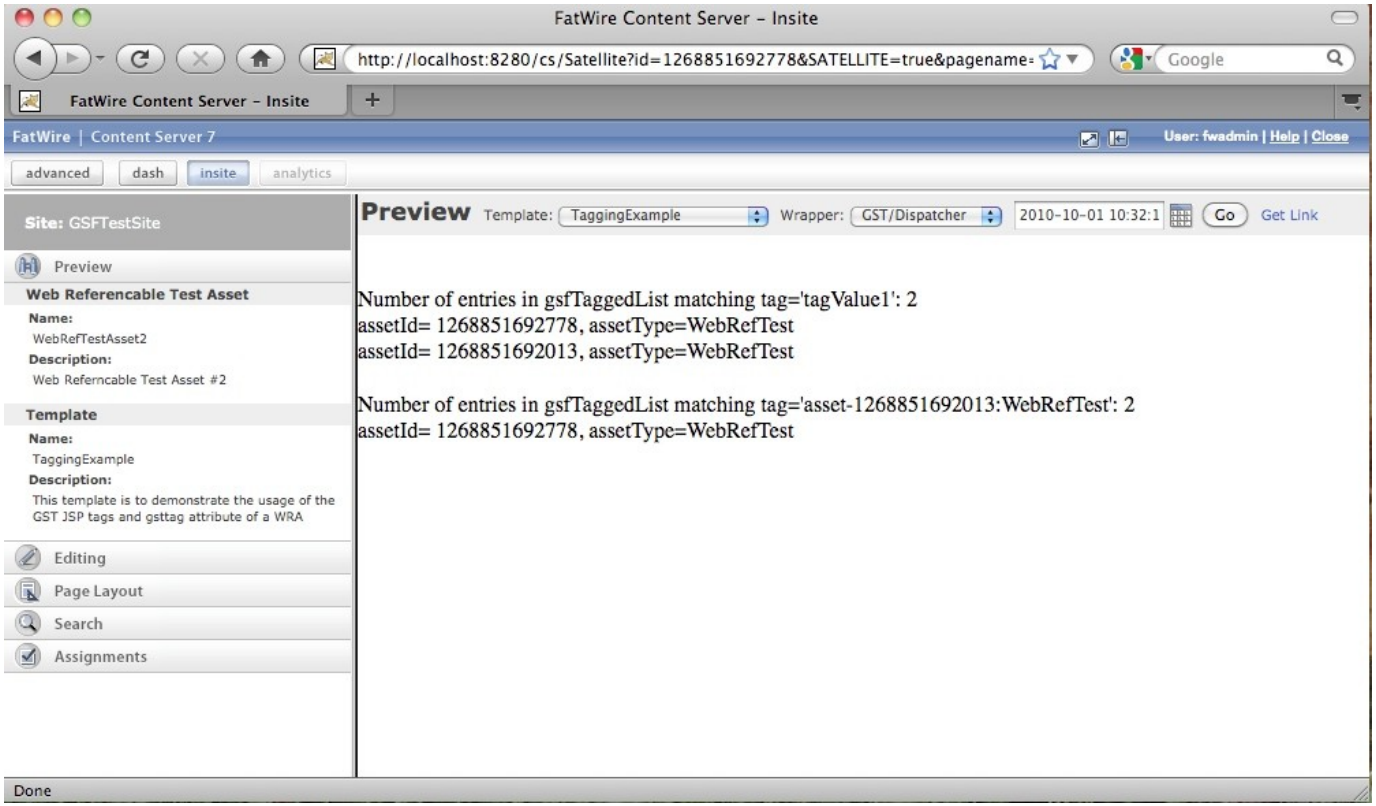
Edit the first Web-Referenceable Asset created (WRA #1) and set its template attribute to WebRefTemplateGeneric

Edit the second Web-Referenceable Asset createtd (WRA #2) and set its template attribute to WebRefTaglibExample

Preview WRA #1 with template set to WebRefTemplateGeneric. Yyou should get something like:



Preview WRA #2 with template set to WebRefTaglibExample. You should get something like:


FatWire Content Server - Insite
http://localhost:8280/cs/Satellite?id=1268851692778&SATELLITE=true&pagename=
Google
FatWire Content Server - Insite
FatWire | Content Server 7
User: fwadmin | Help | Close
advanced
dash
insite
analytics
Site: GSFTestSite
Preview
Web Referencable Test Asset
Name:
WebRefTestAsset2
Description:
Web Referncable Test Asset #2
Template
Name:
TaggingExample
Description:
This template is to demonstrate the usage of the GST JSP tags and gsttag attribute of a WRA
Editing
Page Layout
Search
Assignments
Preview
Template:
TaggingExample
Wrapper:
GST/Dispatcher
2010-10-01 10:32:1
Go
Get Link
Number of entries in gsfTaggedList matching tag='tagValue1': 2
assetId= 1268851692778, assetType=WebRefTest
assetId= 1268851692013, assetType=WebRefTest
Number of entries in gsfTaggedList matching tag='asset-1268851692013:WebRefTest': 2
assetId= 1268851692778, assetType=WebRefTest
Done

4 PART IV: APPENDICES

4.1 Install and configure Maven

If you haven't already install Maven onto your machine.

Download from: http://maven.apache.org/download.html

I have it installed in: /usr/share/java/maven-2.2.1/bin/mvn

Note: For Windows you may have to include plexus-utils-1.4.1.jar to the Maven lib directory.

Comment [DD2]: Why is this, I don't recall having to do this?

Comment [ATF3]: I'm not sure… I would like some feedback from the community on this.

You can download this jar from: http://maven.apache.org/shared/maven-shared-jar/dependencies.html

Click the file: plexus:plexus-utils:jar:1.4.1

Dependency Repository Locations

Repo ID	URL	Release	Snapshot	Blacklisted
codehaus.snapshots	http://snapshots.repository.codehaus.org	-	Yes	-
apache.snapshots	http://people.apache.org/repo/m2-snapshot-repository	-	Yes	-
tars-snapshot-repo	http://57.200.1.171:9090/nexus/content/groups/tars-all/	Yes	Yes	-
tars-repo	http://57.200.1.171:9090/nexus/content/repositories/tars-release-repo/	Yes	Yes	-
codehaus-snapshots	http://snapshots.repository.codehaus.org	-	Yes	-
apache-snapshots	http://people.apache.org/maven-snapshot-repository	-	Yes	-
snapshots	http://snapshots.maven.codehaus.org/maven2	-	Yes	Yes
central	http://repo1.maven.org/maven2	Yes	-	-

Repository locations for each of the Dependencies.

Artifact	codehaus.snapshots	apache.snapshots	tars-snapshot-repo	tars-repo	codehaus-snapshots	apache-snapshots	central
classworlds:classworlds:jar:1.1-alpha-2	-	-		-	-	-	
commons-collections:commons-collections:jar:3.1	-	-		-	-	-	
jakarta-regexp:jakarta-regexp:jar:1.4	-	-		-	-	-	
junit:junit:jar:3.8.1	-	-		-	-	-	
org.apache.bcel:bcel:jar:5.2	-	-		-	-	-	
org.apache.maven:maven-artifact:jar:2.0.7	-	-		-	-	-	
org.apache.maven:maven-model:jar:2.0.7	-	-		-	-	-	
org.codehaus.plexus:plexus-container-default:jar:1.0-alpha-8	-	-		-	-	-	
org.codehaus.plexus:plexus-digest:jar:1.0	-	-		-	-	-	
org.codehaus.plexus:plexus-utils:jar:1.4.1	!	!		!	!	!	
Total	**codehaus.snapshots**	**apache.snapshots**	**tars-snapshot-repo**	**tars-repo**	**codehaus-snapshots**	**apache-snapshots**	**central**
10 (compile: 10)	0	0	10	0	0	0	10

Configure Maven: Please follow the Installation Instructions on the site: http://maven.apache.org/download.html