U.S. Government Printing Office
Federal Digital System

System Design Document

Volume III: FDsys Publish

R1C2 Edition

Prepared by: FDsys Program

Office of the Chief Information Officer
U.S. Government Printing Office

September 2008

Revision History

Revision	Date	Description
0.1	4/2/2008	Start initial draft
0.2	4/22/2008	Split FDsys Push component away from higher-level Search Design document, complete initial draft for review
0.3	4/22/2008	Removed <![CDATA[]> wrapping (can be handled by XML), removed Appendix A (more appropriate for another document)
0.4	4/27/2008	Implemented updates suggested by Deng Wu, including sections on security. Also defined .xsl file management, clarified inputs, and added additional command types. Change name from "Push" to "Publish". Added error notification back to Documentum.
0.6	4/28/2008	Added missing "close()" commands to Documentum sample code, revved version to 0.6 (mistakenly distributed 0.4 as 0.5)
0.7	5/5/2008	Remove embargo tags in favor of a simpler, search-side mechanism. Incorporate final comments from Deng Wu
0.8	5/5/2008	Add class and method for converting a package ID to a package file as a reusable utility.
0.9	5/14/2008	Added information on servlet deployment, what "unpublish" means, why do we pull from the CMS (instead of push), added a full-path method to ACPCacheUtility, changed location of DMZ, added method for detecting updates to .xsl files
1.0	5/25/2008	Minor updates from peer review. Also decided to perform all exports to <packagedir>.swp (followed by rename/delete) to reduce down-time for package updates
1.1	7/7/2008	Added note to publish.xml section to reference the ACPCacheUtility class. Added a logging section to describe the logging functionality to implement. Added new tasks section, for tasks determined as implementation occurs.

1.2	8/5/2008	Updated Documentum Metadata requirements section based on discussions with Paul and Documentum developers. Removed documentum server information from publish.xml and added a section “DFC Properties File” under the Documentum Interface section. Replaced references to search.xsl to refer to mods.xsl. Updated Documentum metadata section to reflect new datetime fields used to improve publish performance.
1.3	8/5/2008	Reformat the front-matter for inclusion in the official SDD.

Table of Contents

1. Background

FDsys will have a custom publish and index notification service. The purpose of this service will be to: 1) Extract documents from the Content Management System (Documentum) and transfer those documents to the ACP cache using Documentum APIs, 2) Notify FAST of new documents to index, and 3) Handle deletes as well as updates.

Many other options were evaluated before it was decided to write a custom program for this purpose. Specifically, we considered:

- Using the FAST Documentum Connector
- Incorporating the content publish and index notification by adding modules to the FAST document processing pipeline
- Using the Documentum Site Caching Service to publish documents
- Using the FAST file traverser to identify updates made to the ACP cache

Our analysis showed that none of these approaches would be able to maintain the integrity between the ACP Cache directory and the FAST indexes both for daily processing and when confronted with disaster recovery scenarios.

Only a single program which controlled both the index notification and the document publication process together would be able to maintain integrity and synchronization in all situations. Since the Documentum Site Caching Service appears ill suited to notifying FAST for new index updates, only a special purpose program will fully satisfy system stability and accuracy requirements.

Fortunately the FDsys Publish program is made up of well understood components and will be straightforward to implement and test.

Search Technologies (www.searchtechnologies.com) works as a services partner with FAST Search & Transfer, providing technical and project management personnel in the delivery of search projects. We were the 2006 Alliance Partner of the Year for FAST, having delivered exceptional services value to FAST and FAST clients.

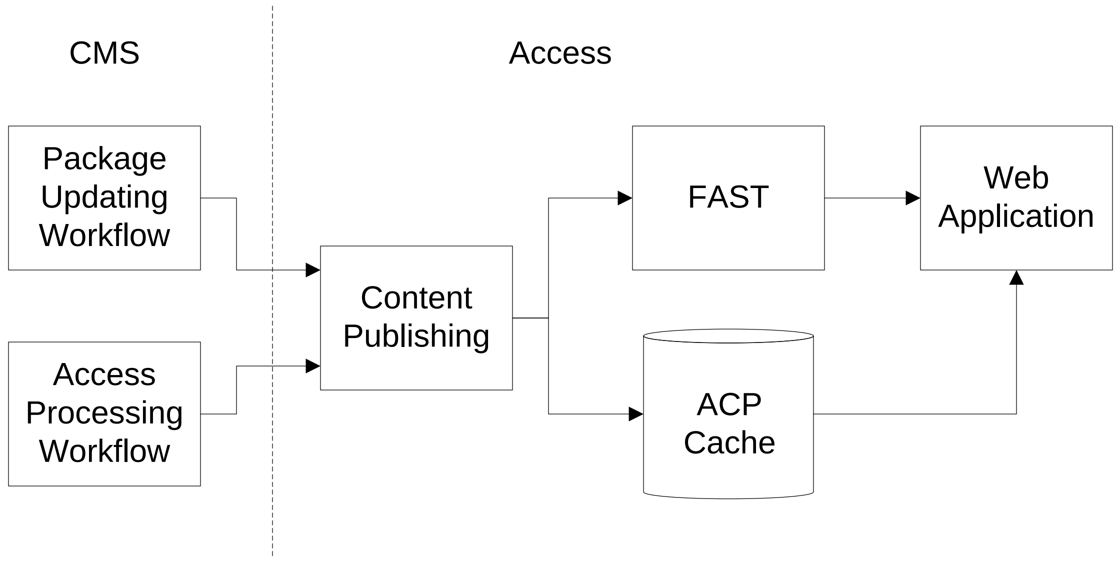
Founded in May 2002 and headquartered in Herndon, VA, Search Technologies is a privately held, profitable provider of comprehensive search solutions based on in-house and third party products. The company operates worldwide, with offices in North America, Central America and Europe. Search Technologies is comprised of many seasoned leaders in our industry, as well as up and coming young professionals – creating the perfect foundation and eye to the future.

Search Technologies focuses on delivering comprehensive search based solutions to enterprise and government customers. We have expertise with FAST's FDS4 and ESP5 product line as well as with their InStream, Unity, ImPulse, and AdMomentum SDA products. In addition to the services delivered to FAST customers, Search Technologies is an active reseller of FAST's products.

Search Technologies also has expertise with many other search engines, including RetrievalWare from Convera and the open source search engine, Lucene. Whatever the needs when it comes to search, Search Technologies is able to deliver experienced personnel.

2. Architecture Overview

The FDsys Publish program is the interface between package workflow and the search engine and ACP cache.



The interface between the workflow and content publishing will be very simple. In order to publish a package, the workflow will need to:

- Set the metadata field "is_published" to true
- Update the metadata field "publish_change_datetime" to the current date and time
- If "first_publish_datetime" is null, then set it to "publish_change_datetime"

Content publishing, when it next runs, will pull all recently updated packages from the Content Management System and push them to FAST and to the ACP Cache.

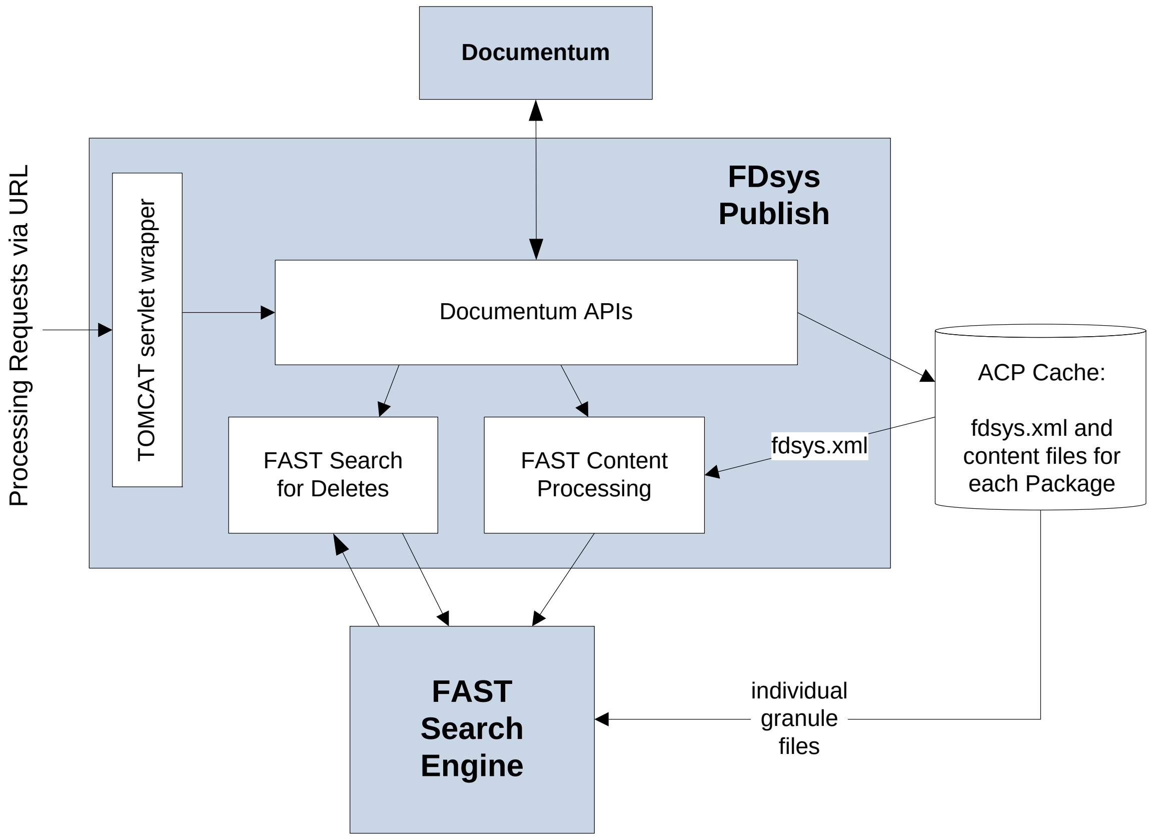
Additionally, workflow can send a URL command to the Content Publisher to immediately publish a package or set of packages.

The FDsys Publish program is divided into the following major sections:

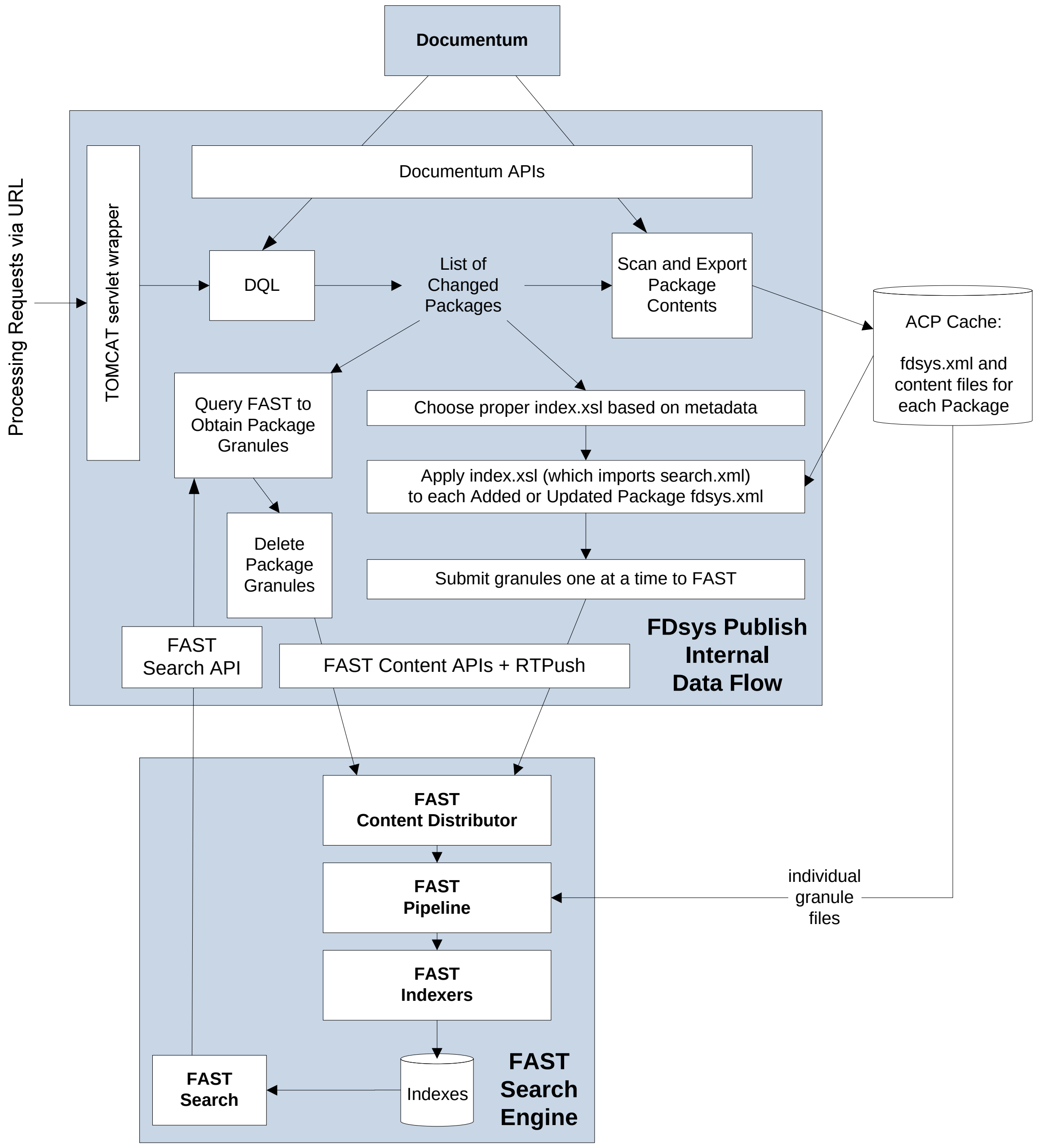
- **Java Servlet Wrapper** - Receives indexing requests via HTTP and processes the requests.
- **Documentum APIs** - The Documentum DFC API will be used to executes DQL (Documentum Query Language) to obtain a list of updated packages that need to be processed and exported to the ACP Cache.
- **FAST Search for Deletes** - When packages are made up of multiple granules, queries FAST to determine the complete list of granules associated with a package so they can be deleted.

- **FAST Content Processing** - Converts the package metadata to FASTXML, divides it up into individual documents and submits them to FAST using the FAST content API.

These components combine together to create the FDsys Publish program as follows:



This next diagram dives down to one level of additional detail to more precisely show how data flows through the FDsys Publish program.


Documentum
Documentum APIs
Processing Requests via URL
TOMCAT servlet wrapper
DQL
List of Changed Packages
Scan and Export Package Contents
ACP Cache:
fdsys.xml and content files for each Package
Query FAST to Obtain Package Granules
Choose proper index.xsl based on metadata
Apply index.xsl (which imports search.xml) to each Added or Updated Package fdsys.xml
Delete Package Granules
Submit granules one at a time to FAST
FAST Search API
FAST Content APIs + RTPush
FDsys Publish Internal Data Flow
FAST Content Distributor
FAST Pipeline
individual granule files
FAST Indexers
Indexes
FAST Search
FAST Search Engine

The Internal Data Flow of the FDsys Publish Program

The components on this diagram are described in detail in the following sections.

2.1. Other Architectural Considerations

Some additional notes on the above architectural diagrams:

- DQL statements are not embedded in the Java Code of the FDsys Publish program. Instead they are located in an external configuration file (publish.xml). This allows the FDsys Publish program to be flexible should there be any changes to the Documentum DQL language in the future, or for additional publishing requirements (such as to republish or re-index an entire collection).
- It was decided to query FAST to determine the complete list of granules which belong to a package (rather than accessing the existing ACP Cache) to improve system reliability. Since it is the FAST indexes which are being updated, it is proper to query the FAST indexes themselves to determine which pieces need to be deleted.

  This way, there can be no discrepancy between the contents of the ACP Cache and the FAST indexes which can not be fixed by simply republishing (or merely re-indexing) the necessary packages.
- "RTPush" is a multi-threaded, connection-pooling wrapper around the standard FAST content APIs available from Search Technologies. "RT" stands for "Real-Time".

2.1.1. "Unpublish"

The FDsys Publish program will be completely responsible for the management and maintenance of both the ACP Cache and the search engine indexes.

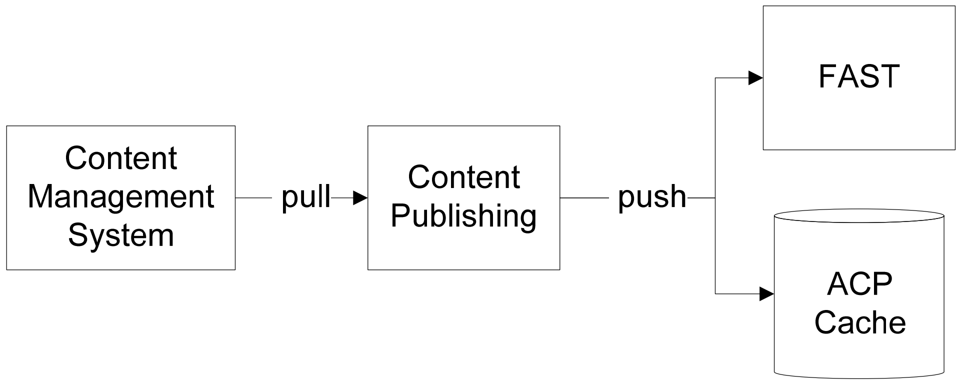
When a package is "unpublished", it will be deleted from the ACP cache and from the FAST indexes and therefore will no longer be available to public users.

However, the package may still be available to authorized users, at the discretion of the archive management workflow.

Examples of packages which need to be "unpublished" include when new star print versions are available or when more complete versions of some packages (such as congressional indexes) are produced.

2.1.2. Pull vs Push

The FDsys Publish application is the interface between the content management system and the FAST search engine and ACP cache. The FDsys Publish program will *pull* data from the Content Management System, and then will *push* that data to the FAST indexes and the ACP Cache, as follows:



It was decided to make the FDsys Publish pull data from the CMS for the following reasons:

- FDsys Publish is responsible for maintaining the FAST indexes and the ACP cache. Therefore, it is in the best position to know the state of those indexes and the additional requirements.
    - For example, if the indexes are corrupted and need to be pulled from backup, FDsys publish will know exactly what date-range of packages will need to be refreshed.
- The FDsys Publish program can pull data from the CMS system at the rate it needs. Therefore, there will be no chance that updates will be lost because the indexers are "too busy" or "unable to keep up".
- The architecture allows for multiple FDsys publish programs to be created to create multiple, parallel sets of indexes from the same Content Management System, should this be required for advanced availability and recovery options.

3. Requirements

This section identifies the technical derived requirements (and the associated system requirements where available) which are satisfied, wholly or in part, by this design.

Note 1: The requirements which are solely the responsibility of the FDsys Publish Program are indicated below with an asterisk (*).

Note 2: For each derived requirement, the parent requirement(s) are identified in square brackets.

3.1. Notification Requirements:

- **Publish**
  [2556, 2557, 2563] FAST must be notified of newly published packages
- **Update**
  [2561] FAST must be notified of any package changes which require the package to be re-indexed

- Metadata changes
- New renditions which need to be indexed

- **Unpublish**
  [2561] FAST must be notified of any package which becomes “unpublished”
  - Updated versions of publicly available documents (e.g. Congressional Index, as new sections of it are added each day)
  - Corrected version of documents (the “starprint”) versions, the old version should be unpublished

The FDsys Publish program will run periodically and will process all documents who have a "publish_change_datetime" which is later than the last time the indexes were updated.

It is the responsibility of CMS workflow to update the "publish_change_datetime" metadata field for packages that need to be published or re-published as per the requirements above.

RD-2556	The system shall provide the capability to search for and retrieve content from the system.
RD-2557	The system shall provide the capability to search for and retrieve metadata from the system.
RD-2561 *	The system shall provide the capability to search content that is currently available on the GPO Access public Web site.
RD-2563	The system shall provide the capability to search and retrieve unstructured content (e.g., text).

* These requirements allocated solely to the FDsys Publish Program.

3.2. Re-index requirements:

- **[39] Re-process all updates since <date/time>**
  There are many failure modes which will require reprocessing of package updates.
  - FAST indexers down
  - FAST indexes corrupted
  - Software bugs
- **[39] Re-build entire index from scratch**
  - For disaster recovery
  - Maybe required when fielding a new version of the FDsys
- **[39] Re-build a publication and/or collection**
  - If parsing for a publication is fixed/improved
  - If the publication gets assigned to a different collection (assuming index-time collection assignment)

RD-39	The system shall support an average peak time availability of 99.7%.

3.3. ACP Cache Maintenance Requirements:

- **Publish**
  - [2417, 2418, 3733, 3740, 3741, 3862] Newly published packages must be copied to the ACP cache
- **Update**
  - [2417, 2418, 2424, 3740, 3741] Updated packages must be similarly updated in the ACP cache
- **Un-publish**
  - [2417, 2418, 3740, 3741, 3943] Any package which becomes “unpublished” must be removed from the ACP cache
- **Selecting**
  - [2326, 2361, 3734] Only in-scope packages will be copied to the ACP cache
  - [2327, 3733] Only public renditions will be copied to the ACP cache

RD-2326	The system shall provide the capability to limit access to content that is out of scope for GPO's dissemination programs.
RD-2327	The system shall provide the capability to limit access to content that has not been approved by authorized users for public release.
RD-2361	The system shall provide the capability for users to access in scope final published versions of ACPs.
RD-2417 *	The system shall provide the capability to manage content that is used for access.
RD-2418 *	The system shall provide the capability to manage metadata that is used for access.
RD-2424	The system shall provide the capability for an existing ACP to be modified.
RD-3733 *	The ACP cache shall only contain public access renditions of final published content and associated metadata
RD-3734	The system shall provide the capability for authorized users to prevent external ACPs from being created from internal ACPs.
RD-3740 *	For all publicly available content, the publicly available content shall be the same as the content available to authorized users.
RD-3741 *	For all publicly available metadata, the publicly available metadata shall be the same as the metadata available to authorized users.
RD-3862	The system shall provide access to digitally signed PDF content in public user search results.
RD-3943	The system shall delete an ACP in the case that the content has been replaced by a star print version.

* These requirements allocated solely to the FDsys Publish Program.

3.4. ACP Cache Re-build requirements:

- **[39] Re-process all updates since <date/time>**
    - If the disk space fills up and updates get lost
    - If the update mechanism down for other reasons
- **[39] Re-build entire ACP Cache from scratch**
    - For the initial build
    - For disaster recovery

RD-39	The system shall support an average peak time availability of 99.7%.

3.5. Processing Requirements:

- [3756, 2738] When a package is published, all granules need to be extracted and indexed
- [3756, 2738] When a package is un-published, all granules need to be deleted
- [3756, 2738] When a package is updated, all updates need to be processed (including an update which results in more or fewer granules)
- [3756] The proper rendition will need to be indexed for each granule
- [662] Each granule needs to be indexed with it's own metadata, as well as the merged metadata of all it's parents
- [2565, 2566, 2567] Metadata will need to be mapped to FAST index-profile fields to support all search features
    - Fielded searching
    - Search Results and Content Details
    - Document access
    - Package Table of Contents and Collection Browsing
    - Complex metadata searches (via the "Search Schema")

RD-662	The system shall allow GPO to define the level of granularity that content can be retrieved at.
RD-2565	The system shall provide the capability to search and retrieve semi-structured content (e.g., inline markup).
RD-2566	The system shall provide the capability to search and retrieve structured content (e.g., fielded).
RD-2567	The system shall provide the capability to search for content by means of querying metadata.
RD-3756	The system shall provide the capability for users to search for granules.

RD-2738	The system shall provide the capability to return search results at the lowest level of granularity supported by the content package.

3.6. Status Monitoring:

- [1356] For the purposes of debugging and monitoring, the status of packages being processed by the FDsys Publish program should be available for inspection

RD-1356	The system shall have the capability to monitor real-time performance of the system in terms of service levels.

4. HTTP Servlet Wrapper

The purpose of the HTTP servlet wrapper is to receive processing requests from HTTP clients. Such an architecture will allow for processing requests to come directly from the authorized user's workstation, if (for example) they wish to republish a modified package immediately.

The HTTP Servlet Wrapper further gives the control of exactly when to publish packages to the Documentum Servers. It is expected that authorized users will be provided with a "Publish Now" button which will send the appropriate HTTP command to the FDsys Publish program to immediately download and process a specified package.

Further, Documentum could be used to manage the scheduled jobs as well, such as the periodic "update" commands. This would allow, for example, Documentum Workflow to delay a scheduled publish event if (for whatever reason) the content was not ready to be published.

4.1. HTTP Commands

These requests will be handled with the following HTTP GET commands:

Description	HTTP Get Arguments	Example
Publish Package	package=<packageid> export=<yes/no>	http://host:9001/publish?cmd=publish &dql=package&packageid=fr01no06
Publish Date Range	from=<date/time> to=<date/time> ("to" is optional) export=<yes/no>	http://host:9001/publish?cmd=publish &dql=range&from=2008-04-00T00:30:00& &to=2008-04-04T00:30:00
Publish Updates	<none>	http://host:9001/publish?cmd=update
Set Update	datetime=<datetime>	http://host:9001/publish?cmd=setupdate &datetime=2008-04-04T00:30:00

Process Other DQL	dql=<dql-name> export=<yes/no> <param1>=<val1> <param2>=<val2> . . .	http://host:9001/publish?cmd=publish &dql=reindexcollection&export=no&collection=fr
Status	<none>	http://host:9001/publish?cmd=status

Notes:

- Each command will return an XHTML page as a response. The resulting page will identify the status of the command, and will provide additional feedback in case of failure (i.e. exactly what failed).

  Note that for commands which execute DQL statements, in the current design the response will come back after the DQL statement has completed executing.
- Commands to provide system status will be executed immediately and will return an HTML file which describes the results.
- Commands for publishing packages will perform the DQL statement to access the package ID and publish-specific metadata. The resulting list of packages will be stored on a queue for background threads to process.

Tasks:

T1. Create the HTTP Servlet wrapper with a doGet() call inside an instance of Tomcat running on the FAST admin node.

T2. Parse the HTTP Get arguments and return an HTML response for each.

4.1.1. The "package" command

Immediately process a single package. How the package is processed (i.e. is it added, updated, or deleted) will be determined based on the package metadata retrieved from Documentum.

Name	Allowed values	Description
packageid	FDsys ACP Package ID	The package ID of the package which should be immediately published
dql	"package"	Specifies the DQL statement to be used to implement the package command. Must always be "package"
export	"yes" or "no"	Should the package be exported from Documentum and written to the ACP Cache? If "no", only the FAST index notification is performed. If missing,

		defaults to "yes"

Note:

- The purpose of "export=no" is to allow the system to re-index content to handle situations (especially in development), where the data from the fdsys.xml metadata file is reorganized before it is indexed. This could be the case if there are new FAST ESP index-profile fields, or changes to the index.xsl or mods.xsl transformation files.

4.1.2. The "range" command

Process all packages which have a "publish_change_datetime" within the specified datetime range.

Parameters:

Name	Allowed values	Description
from	Date in ISO 8601 format	The starting date and time of the date range.
to	Date in ISO 8601 format	The ending date and time of the date range.
dql	"range"	Specifies the DQL statement to be used to execute the publish range command. Must always be "range".
export	"yes" or "no"	Should the package be exported from Documentum and written to the ACP Cache? If "no", only the FAST index notification is performed. If missing, defaults to "yes"

4.1.3. The "update" command

Process all updates since the last time the "update" command was run.

The update command has no parameters. All packages updated since the last time it was run will be exported from Documentum to the ACP cache and reindexed into FAST. Behind the scenes, the "update" command will execute the "range" DQL statement.

4.1.4. The "setupdate" command

Set the "last update" time. This is usually performed after the initial bulk load, to specify the time from which incremental updates are to begin. Note that if the "datetime" argument to this command is missing, it sets the last update time to the current date and time.

Name	Allowed values	Description
datetime	Date in ISO 8601 format	The date and time to which the "last update" time should be set

Note:

- The date specified to the "setupdate" command will be stored in a local file for use by the FDsys Publish program only. See section 9 for more details.

4.1.5. The "other" command

Executes any other DQL statement that has been pre-configured into the configuration file. Any other set of parameters may be provided which are substituted into the DQL statement as needed. The "other" command allows for other publishing and disaster recovery needs to be implemented which may not have been anticipated at the time this document was written.

Note that the "other" command does *not* take raw DQL as an argument. It will only execute DQL statements that have been preconfigured into the publish.xml configuration file. These DQL statements are referenced by name.

Parameters:

Name	Allowed values	Description
dql	The name of a DQL statement from the publish.xml configuration file	Specifies which DQL statement should be executed.
<param1>, <param2>, . . .	As appropriate to the parameter	Identifies a parameter to be substituted into the chosen DQL command. The exact name and value of the parameter will depend on the DQL command being executed
export	"yes" or "no"	Should the package be exported from Documentum and written to the ACP Cache? If "no", only the FAST index notification is performed. If missing, defaults to "yes"

4.1.6. The "status" command

Get status on all of the packages currently being processed by the FDsys Publish component. See section 12 for more details

No additional parameters are required for the "status" command.

4.2. Admin User Interface

A simple HTML Admin User Interface will be provided which allows admin users to execute HTTP commands directly. All of the commands identified in section 4.1 will be available to be executed.

The purpose of the Admin User interface is strictly for system testing, status, and disaster recovery scenarios. It is expected that Documentum will be responsible for executing HTTP commands during normal usage.

For security purposes, the Admin User interface will require a simple log-in screen before it can be accessed. The username and password entered will be validated against the standard FAST Admin Accounts database, using the FAST Admin APIs.

Tasks:

T3. Program the Admin User Interface using a simple HTML user interface configured to run in the same Tomcat instance as the FDsys Publish servlet.

T4. Require that users enter a username and password before they are allowed access to the Admin User Interface. Verify the username and password against the standard FAST admin accounts provided through FAST Home, using the standard FAST Admin APIs.

4.3. HTTP Security

It is important for the FDsys program to only be accessed by authorized users and/or servers. This will be accomplished using SSL with a client-authenticated certificate. In this protocol, the FDsys Publish program will demand a certificate from the HTTPS client, to ensure that only authorized clients are able to send commands to Publish.

The certificate will be generated with the Java "keytool" program and the certificate will be transferred and accessible only to the Documentum servers. When needed, a WebTop user interface will be configured to use the certificate to initiate an SSL connection to the FDsys Publish program to execute the desired command.

Tasks:

T5. Generate the certificate and test SSL with client authentication to the FDsys Publish server.

4.4. Deployment

Since the purpose of the FDsys Publish program is to maintain and manage the FAST indexes and the ACP cache, it will be deployed into the same servlet container provided by the FAST search engines for administration and other search engine services.

This will centralize management of the data and indexes required for search into a single location, so they can be properly managed and maintained together.

T6. Configure the FAST servlet container for the FDsys publish application. Create the necessary FAST deployment scripts for deploying the FDsys publish application after FAST has been installed.

5. Configuration

5.1. publish.xml

A special XML configuration file will hold configuration information for the FDsys Publish program. It is recommended that the Apache "Digester" class (http://commons.apache.org/digester) be leveraged for this task.

An example of publish.xml is as follows:

```
<fdsys-publish>
  <dql command="package">...DQL statement goes here...</dql>
  <dql command="range">...DQL statement goes here...</dql>
  <!-- Additional DQL statements can be created here for other
       publishing and/or index notification requirements -->
  <collection-config dir="/FDsys/etc/collections"/>
  <acp-cache>/smnt1/fdsys/ACP</acp-cache>
  <last-updated-file>updated.dat</last-updated-file>
  <max-threads n="4"/>
  <documentum>
    <docbase>GPORespostory</docbase>
    <user>fdsyspublish</user>
    <password>changemeplease</password>
    <error-folder>/FDsys Publish/errors</error-folder>
  </documentum>
  <fast>
    <index-servers>fastnode1.gpo.com:16100, fastnode2.gpo.com:16100
    </index-servers>
    <query-servers>fastnode1.gpo.com:15100, fastnode2.gpo.com:15100
    </query-servers>
  </fast>
</fdsys-publish>
```

This data will be loaded into the FDsys Publish program on startup. The currently defined configuration elements are as follows:

<dql command="package">

> The DQL statement used to fetch a single package based on the package ID (see sections 5.1.1 and 8 for more details).

<dql command="range">

> The DQL statement used to fetch a set of packages to be published based on a from and to date range (see sections 5.1.1 and 8 for more details).

<collection-config dir="">

> Specifies the directory where the collection configuration files are located. For FDsys Publish, this is the directory where the index.xsl and mods.xsl for each collection can be located.

<acp-cache>

The file system location where the ACP Cache is located. The ACPCacheUtility class should be given this value (see section 6.1.1 for more details).

<last-updated-file>

The file path where the last updated date/time is written. This file is used for the "update" command, so that FDsys Publish will know the start-time to use for selecting which documents to publish.

<max-threads n="4"/>

Specify the maximum number of simultaneous processing threads should be spawned to handle package processing.

<documentum>

A collection of parameters all related to the Documentum Server. Comprising of:

<docbase>

The Documentum repository which contains the ACP packages to be published and indexed.

<user>

The Documentum user name to use to log into Documentum.

<password>

The Documentum password to use to log into the Documentum servers.

Note: The Documentum user and password may be removed from the configuration file (or an additional parameter added), if we decide to use a "ticket granting" method for login. To be determined.

<error-folder>

The Documentum folder to store any error messages which occurred during processing of commands. See section 7.3 for more details.

<fast>

Holds information required to connect to the FAST search servers. Specifically:

<index-servers>

A list of possible FAST Content Distributor servers which can receive index notifications via the FAST content APIs. Note that multiple servers are specified her for process failover in the event that one server is down.

<query-servers>

A list of possible FAST QR servers which can process search requests. Again, multiple servers are specified here to allow for load balancing and failover by the FAST search APIs.

Tasks:

T7. Open up the publish.xml file on startup and load all configuration data into memory.

T8. Every time an HTTP command is executed, check the touch-datetime on the publish.xml. If it is more recent than the last time it was loaded, then reload the publish.xml file into memory.

5.1.1. DQL Statements

The DQL statements specified in the publish.xml file can contain substitutable parameters.

For example:

```
select package_id, r_object_id, ccode, first_published_date,
       publish_change_date, fdlp_in_scope, is_published, is_purged
from dm_folder
where publish_change_date >= DATE('${from}') and
      publish_change_date < DATE('${to}')
```

(Note that this DQL is only a sample and may have syntactic or semantic errors)

In this example, ${from} and ${to} are substitutable parameters which will be replaced with date strings.

The substitutable parameters come from the HTTP command parameters which were used to execute the command, and have the same names as the command parameter names.

See section 8 for more details.

5.2. index.xsl and mods.xsl

The FDsys Publish program does not contain any special software for mapping from FDsys metadata fields into FAST ESP index-profile fields.

Instead, two XSL Transformation files will perform this task on a collection by collection basis:

mods.xsl	This transformation will create the XML document that is indexed into the "xml" index-profile field for FAST scope searching. FAST scope search allows users to create complex queries directly over complex structured XML data. See the "Search Schema Definition" document for more details on the format produced by mods.xml.

index.xsl	This transformation file maps the metadata from the ACP into all of the other FAST index-profile fields (such as the title, URL, abstract, frompage, etc.). The output of index.xsl will be in FASTXML format. This format is defined by FAST for easy processing by the FAST ESP document processing pipeline. The mods.xsl will be called out from within the index.xsl program using the <xsl:import> element.

See the "ESP File Traverser Guide" for more documentation of the FASTXML format.

See 'Appendix D: Sample "index.xsl" ' and 'Appendix E: Sample "mods.xsl" file' for examples of "index.xsl" and "mods.xsl" transformations.

5.2.1. Locating XSL Transformations Files

There will be a different index.xsl and mods.xsl for each different collection to be processed in FDsys. These files will be located in a common directory, such as:

```
/FDsys/collections/<collection-name>/
                                    index.xsl
                                    mods.xsl
```

So, for example, to locate the "index.xsl" file for the Federal Register (code = FR), the FDsys Publish program will load the "/FDsys/collections/FR/index.xsl" file.

Having all collection-based information in a common directory makes it easy to maintain configuration control and deployment for new collections. A new collection can be installed by merely dropping in a directory of the necessary configuration and XSL transformation files into the proper location, without tweaking any other "master" configuration file.

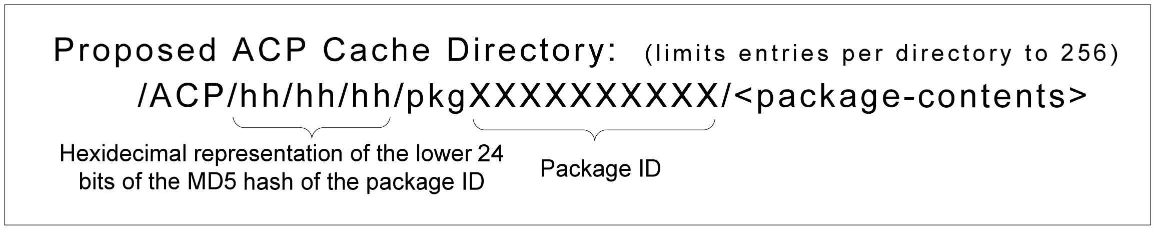
Tasks:

T9. Create a routine to locate, compile, and cache the index.xsl transformation as needed for transforming metadata from the ACP fdsys.xml file into FASTXML.

T10. Every time a document is loaded, check the index.xsl last-modified date time. If it is more recent than the last time the template was loaded, reload the .XSL file.

6. ACP Cache Structure

The structure of the ACP cache will be as follows:

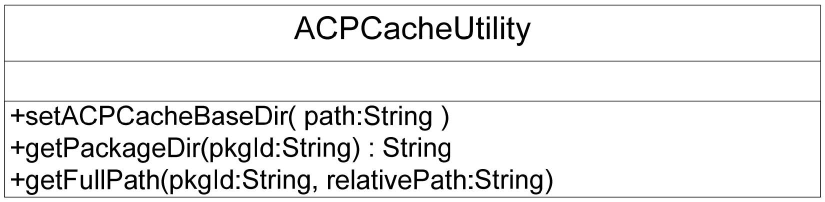

Proposed ACP Cache Directory: (limits entries per directory to 256)
/ACP/hh/hh/hh/pkgXXXXXXXXXX/<package-contents>
Hexidecimal representation of the lower 24 bits of the MD5 hash of the package ID
Package ID

The purpose of this structure is to limit the number of entries per directory in the ACP cache to a maximum of 256 and to allow for growth to billions of possible packages.

The package directory structure will be created on-the-fly as each package is exported from Documentum. Only sub-directories which contain packages (or once contained packages) will exist in the directory structure.

6.1.1. ACP Cache Utility

Since the function to convert a package access ID into a package directory name will be needed by many different system components, a utility class will be created to do this:



```
ACPCacheUtility

+setACPCacheBaseDir( path:String )
+getPackageDir(pkgId:String) : String
+getFullPath(pkgId:String, relativePath:String)
```

Method	Description
void setACPCacheBaseDir(String path)	Set the base directory where the ACP cache is located. This will be stored in a static variable.
String getPackageDir(String pkgId)	Convert a package ID into the full path name for the package directory. Produces a path in the format described above.
String getFullPath(String pkgId, String relativePath)	Convert a package ID and a relative path into a full path name. The full path identifies the location of the file on the local machine so it can be accessed directly and (for example) streamed back to the user.

T11. Create the utility class which converts a package access ID to an ACP directory path, using the MD5 of the package access ID, as described above.

7. Package Processing Overview

The steps for processing a package are as follows:

1. Based on the type of command received by the Tomcat Servlet wrapper, look up the proper DQL command from the publish.xml configuration file. Substitute the variable parameters from the HTTP command into the DQL statement and execute it using the Documentum DFC APIs.
2. Use the flags (is_published, is_public, is_purged) from the DQL statement to determine if the package is an update and/or delete.

   Updated packages are determined by comparing the "first_published_datetime" to the "publish_change_datetime". If they are equal, then this is the first time the document has been published (it is an ADD). If they are different, then this is an update.

   Deleted packages will have "is_published" or "fdlp_in_scope" set to "false", or "is_purged" set to true.

   For updated or deleted packages:

   a. Use FAST search to find all of the granules contained within the package and delete them from the FAST indexes (see section 10).

   b. (For deletes only, not updates) Remove the package (and all of its contents) from the ACP cache.
3. Export the package from Documentum and load it into the ACP Cache.

   a. Create the package directory (including parent directories, as necessary).

      i. Export the package into <packagedir>.swp

   b. Scan through the contents of the package folder, exporting files and creating sub-directories as necessary.

   c. Repeat step b, for all nested sub-directories, as necessary.

   d. If <packagedir> already exists, rename it to <packagedir>.old

   e. Rename <packagedir>.swp to <packagedir>

   f. Remove <packgedir>.old (and all of its contents) from the ACP cache.
4. For every package to be added or updated, perform the following steps:

   a. Using the document type from the DQL statement, look up the index.xsl file and parameters to be used from the FDsys process-config module.

   b. Apply the index.xsl file from step "a" above to the fdsys.xml file downloaded with the package. The result will be a FASTXML file to index.

c. Due to limitations in FAST, some additional processing on the FASTXML will need to be implemented before it can be submitted to the FAST search engine:

    i. Split the FASTXML file into individual documents based on the <document> tag. Each document will need to be submitted individually to the FAST indexes.

d. Submit each <document> to FAST using the RTPush content API wrapper.

e. Monitor content API callbacks.

    i. If there is an error, retry the document three times.

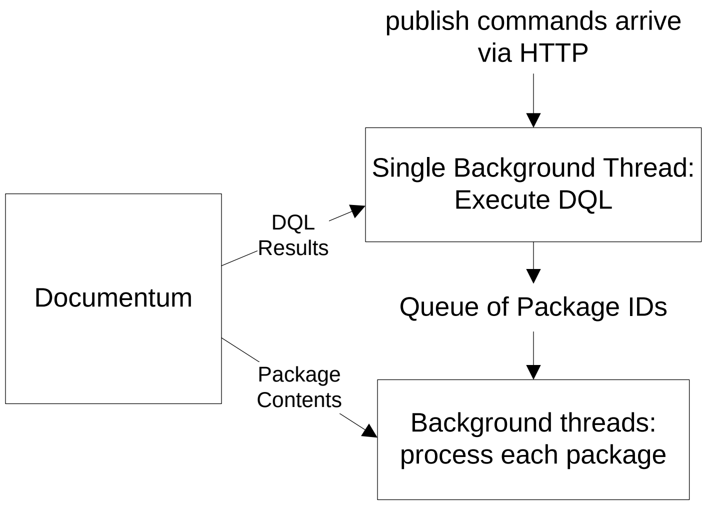
    ii. If all three attempts fail, then log the error in a log file and return an error code to a Documentum Queue.

The tasks required to implement each of these steps are identified in the following sections.

7.1. Processing Threads

The FDsys Publish program will maintain background threads to process package IDs. Multiple threads will be necessary for large batch updates, where many packages will need to be published.

The processing will be divided as follows:


publish commands arrive via HTTP
Single Background Thread: Execute DQL
DQL Results
Documentum
Queue of Package IDs
Package Contents
Background threads: process each package

The servlet which responds to the HTTP command will be responsible for executing the DQL statement to Documentum and building a queue of Package IDs. Each item in the queue will also contain the necessary package metadata (publish_change_datetime, is_published, etc.) required to determine if the package needs to be deleted, added, or updated.

Background threads will then process each package from the queue one at a time. Each background thread will be responsible for deleting the package from the ACP cache, downloading the new package contents, removing the package from the FAST indexes, preparing the package for index processing, and submitting the package granules for indexing.

Finally, for large batch loads, a main() method will be provided so that FDsys Publish can be executed from the command line. This will be for disaster recovery and initial batch load situations where the number of documents returned by the DQL statement is very large.

T12. Spawn background threads when the servlet is initialized. Background threads wait for packages to be put on a package queue for processing. The number of threads to create is specified in the publish.xml configuration file.

T13. Implement a command-line version of the FDsys Publish program for large batch loads.

7.2. Purge

It is assumed that, when a package is purged, that a placeholder for that package will remain in the Content Management System. This placeholder will only need to have a very few metadata values: Package ID, publish_change_datetime, first_publish_datetime, and "is_purged".

It is assumed that this placeholder will never be removed. Therefore, if a package needs to be purged, it should have an "is_purged" flag set to "true", and have all of it's content removed.

It is important to leave placeholders for purged packages in the CMS to ensure that FAST can properly recover from backups. Otherwise, if packages were to be fully removed and then (as part of disaster recover) the FAST indexes were to be recovered to a state before the purge, then the purged package would show up in search results as an orphan.

7.3. Error Handling

Errors in processing content should happen rarely, if at all. And when they do occur (if, for example, disk space is full), it is likely that they will occur for all documents.

Therefore, errors will not be processed on a document-by-document basis, since this will likely result in many hundreds of duplicate messages. Instead, all errors will be aggregated on a per-command basis.

Once a command has completed, any errors accumulated by the command will be written back as a message to a Documentum Queue, so that the appropriate authorized users can be notified.

To be determined: The exact format of the message to be returned.

Tasks:

T14. If errors have occurred while processing an HTTP command, accumulate all of the errors into a message and write it back into a Documentum queue provided for this purpose.

8. Documentum Interface

We will be using the Documentum "DFC" API for accessing packages from the Documentum Docbase. See "Appendix A: Code Fragment for Documentum APIs" for an example which illustrates the Documentum API calls required to process packages in FDsys.

The basic outline for the Documentum portion of the FDsys Publish program is as follows:

1. Initialize the Documentum client, session manager, and session.
2. Execute the appropriate DQL command. The DQL commands will have been pre-loaded from the publish.xml configuration file.
    a. The package ID or date range parameters will need to be substituted into the DQL statement where appropriate.
    b. Any string substituted into a DQL statement should have special characters (such as quotes, etc.) properly escaped.
3. For each package, check its metadata information to determine if the package will need to be added, updated, or deleted.
4. For updates or adds, the package folder will need to be exported from Documentum.
    a. Create the folder in the ACP cache.
    b. Scan through the contents of the folder.
        i. If the item is a sub-folder, check that the folder "is_public", and if so start again at step "a".
        ii. If the item is a document, export the document to the ACP Cache to the appropriate location.

Tasks:

T15. Execute the DQL commands called out by the HTTP command. Substitute the ${parameters} in the DQL statement as necessary. Ensure that special characters in the parameter value (such as quotes) are properly escaped as necessary.

T16. Store the results of the DQL in an in-memory queue to be handled by background threads.

T17. For each item identified by T15, determine if it is an update, delete or add. For deletes and updates, pass the package ID's to "Fast Search and Delete" (see section 10.

T18. For packages to be deleted or updated, physically remove the package (and all nested directories and files) from the ACP cache.

T19. For packages to be updated or added, download the package contents from Documentum and write the package to the ACP cache.

T20. Pass the package IDs of packages to be updated to "FAST Index Notification".

8.1. Documentum Metadata Requirements

Packages accessed by the FDsys Publish program must have certain metadata to ensure proper processing. Specifically, the DQL statement for processing packages must return the following Documentum metadata information:

- **“package_id”** - The ID used to access the data in Documentum.
- **“access_id”** - The ID used by the system outside of Documentum to access the package.
- **“r_object_id”** - The Documentum folder ID for the package folder, so that the folder contents can be examined using the DFC APIs
- **“collection”** - The document collection code (used to locate the proper directory which contains the "index.xsl" and "mods.xsl" files)
- **“package_state”** – Used for the WHERE clause in DQL statements to identify all packages that are packages potentially available for exporting to the ACP Cache. State will be set to ‘ACP’.
- **“is_public”** - Flags to determine if a folder is public.
- **"first_publish_datetime"** - The datetime when the package was first published. This can be compared to "publish_change_datetime" to determine if the package is an ADD (both datetimes are the same) or an UPDATE (the datetimes are different).
- **"publish_change_datetime"** - Used for the WHERE clause in DQL statements to identify all packages which have changed in any way that might require some re-publishing of some sort.
- **"mods_change_datetime"** - Identifies if there is a metadata change. If this datetime is within the datetime of the index-datetime-range, then download a new fdsys.xml and re-index the entire package, Since all mods.xml is copied into the indexes, then a change to the mods will require a re-index. Further, since all fdsys.xml descriptive metadata is stored in mods, then a change to the mods.xml automatically means that the fdsys.xml has changed and needs to be re-downloaded.

- **"content_change_datetime"** - Identifies that a content file has changed. If this datetime is within the datetime of the index-datetime-range, re-export all content files and the fdsys.xml and rebuild the entire package in the ACP Cache. This flag should only be set if an isPublic rendition is changed.
- **"premis_change_datetime"** - Identifies that the premis.xml has changed. If this datetime is within the datetime of the index-datetime-range, download the new premis file ***only***. This does ***not*** require re-indexing or re-exporting the fdsys.xml or content files.
- Any flags which are necessary to determine if the package is to be published. Specifically: **"scope"**, **"is_published"**, and **"is_purged"**

8.2. Publishing A Document

When the CMS needs to publish a document, it will need to perform the following steps:

1. Set the "publish_change_date" to the current date and time.
2. If the "first_published_date" is NULL, then set "first_published_date" to the same date and time, otherwise leave it untouched.

And that's all! On the next update cycle, FDsys Publish will automatically publish and process all packages with a "publish_change_date" younger than the last time the Publish was executed. The "first_published_date" will be used to determine if this package is an "update" or an "add".

8.3. Scheduled vs On-Demand Processing

The FDsys Publish program is a passive program which responds and executes to commands from the outside.

It is expected that there will be two types of commands:

1. Scheduled Commands - running on a periodic basis (e.g. once a night) for processing all of the packages received that day.
2. On-demand Commands - for initiating a publish and/or index notification immediately.

8.3.1. Scheduled Commands

FDsys Publish will not be responsible for initiating scheduled commands. These can be implemented either by the operating system with a CRON job on the local machine, or possibly through a Documentum scheduled nightly job.

8.3.2. On-Demand Commands

Can be initiated either by the Admin User Interface provided with FDsys Publish, or by any other (authorized) HTTP client. For example, an on-demand publish could be performed as part of Documentum work-flow, to immediately process any package which has been manually corrected and whose corrections must be made available to the public as quickly as possible.

8.4. DFC Properties File

For FDsys Publish to be able to connect to Documentum, a dfc.properties file will be included in the installation classpath.

An example of dfc.properties is as follows:

```
dfc.docbroker.host[0]=cms1.dev.fdsys.gpo.gov
dfc.docbroker.port[0]=1489
dfc.globalregistry.repository=R1C_Dev_01
dfc.globalregistry.username=dm_bof_registry
dfc.globalregistry.password=2dWUCxhGpJ8YW10is0v/PA\=\=
```

9.   The "update" command

The update command is the same as the date-range command, except that the from-date is taken from the "last-update-file" file (specified in the publish.xml configuration file), and the to-date is the current date time.

Once the update command has been completed successfully, the time used for the "to-date" will be written to the "last-update-file".

Tasks:

T21. Open the last-update-file and fetch the last update time.

T22. Time stamp the current time, and then process the date range as normal.

T23. Once complete, write the previously stamped time (from T22) back to the last-update-file.

T24. Implement the "setupdate" command, which simply sets the time stamp in the "last-update-file" to the specified time stamp, or to the current datetime if no time stamp has been specified.

10. FAST Search and Delete

Many of the documents to be indexed for FDsys are split into pieces, called "granules". For example, each issue of the Federal Register will be split into (roughly) 150 pieces, one for each regulation or notice which is printed that day.

But publishing updates, deletes, and adds of documents are at the package level, not the granule level. Therefore, in order to ensure that there are no orphan granules, the FDsys Publish program must search for all granules and delete them, before deleting or updating a package.

It was decided to go directly to the FAST indexes to perform this function, rather than going to an older copy of the package on the ACP cache. This will ensure that the FAST indexes are correct, whereas the package on disk may or may not be correct (depending on the disaster recovery scenario).

The following tasks will need to be performed for any package which is deleted or updated. We will use the standard FAST search and content APIs to perform these tasks.

Tasks:

T25. Use the package ID to perform a FAST search to determine the complete list of granules which have been indexed into that package.

T26. Send a "delete" command to the FAST indexes for each granule returned by the search engine.

11. Indexing Packages

As described in section 7, each package will require some special processing before it is sent to the FAST search engine for indexing. This special processing is as follows:

1. Convert the fdsys.xml file to FASTXML. This step maps the fielded data from the fdsys.xml XML <entity>s to FAST index profile fields using the index.xsl template.
    a. Note that the .xslt template which does this conversion will import a second "mods.xsl" template to create the nested "search.xml" file used for complex metadata searches in FDsys. The results of the "mods.xsl" template will be indexed into the FAST "xml" index-profile field.
2. The FASTXML will contain multiple documents, one for each granule that needs to be indexed.
    a. Due to a limitation in FAST, The index notification program will need to split apart these documents and send them one-at-a-time to FAST.
3. The index.xslt template program will put the search.xml inside the <value> entity for the "xml" index-profile field, wrapped with a <![CDATA[ ]]> tag.

We will be using the "RTPush" content API wrapper for sending documents to FAST (available from Search Technologies). This wrapper provides content session pooling and a simplified interface for submitting documents.

An example of the fdsys.xml file can be found in "Appendix B: fdsys.xml Example."

An example of the FASTXML file which is used as input to the FAST indexers can be found in "Appendix C: Sample FASTXML file."

See 'Appendix D: Sample "index.xsl" ' for a sample "index.xsl" program which maps metadata fields from the fdsys.xml file into the FASTXML format.

See 'Appendix E: Sample "mods.xsl" file' for a sample "mods.xsl" program which maps fields from the fdsys.xml file into "Search Schema" format – for feature-rich and intuitive searching.

Tasks:

T27. Load the fdsys.xml file for the package into a Java structure (such as an "InputSource").

T28. Based on the package "collectionCode" attribute, determine the appropriate 'index.xsl' from the FDsys "process-config" module.

T29. Apply the index.xsl template program to the fdsys.xml file to create the index.xml file.

T30. Split up the index.xml file into pieces and then submit each piece for indexing to FAST.

12. Status

A special status page is recommended to provide status on the current indexing and notification progress of the FDsys Publish program. The response to a status command will be an HTML page which displays the status of all outstanding packages.

Packages which have completed processing will not be displayed on the status page.

The possible status values for packages could be:

- Waiting for processing - The package is still in the package queue and has not yet been picked up by a background thread.
- Downloading from Documentum - The package contents is currently being downloaded from Documentum.
- Processing Deletes - The package ID is currently being searched to locate the package granules and the granules are being submitted for deletes.
- X number of Granules Waiting for deletes - Granule deletes are currently being processed by FAST for the package.
- Index Preparation - The package is being prepared for indexing.
- X number of Granules Waiting for Document Processing - Granule updates or adds are currently being processed by the FAST document processors.
- X number of Granules waiting for indexing - Granule updates or adds are currently being written to the FAST indexes.

All of this information should be readily available by inspecting the package queue, the thread objects, and the RTPush connection callback hash tables.

Should this task prove to be too difficult or time-consuming to implement, a simpler version could be implemented where the threads simply write their status to a log file as they complete each task.

Tasks:

T31. Implement the status command. Inspect the various queues, thread objects, and hash tables to determine the status of each package ID. Return the results to the browser as a formatted HTML page.

13. Logging

Using log4j, the FDsys Publish program will log the following:

1. Startup status. This will include connection to FAST and Documentum. Status will indicate success or failure, with a reason if failure occurred.
2. Commands that are received, either via the Servlet or the command line application.
3. For each command received, the package IDs that are affected.
4. For each affected package, the action that occurs. Actions will be add, delete or update.
5. For each package, the result of the action. The result will indicate success or failure, with a reason if failure occurred.

Tasks:

T32. Implement logging in the FDsys Publish program, as described above.

14. Additional Tasks

T33. Map documentum granule files to collection specific filenames inside the ACP cache, using based on mapping rule in configuration files for each collection. The configuration files should reside collection-config dir defined in publish.xml.

T34. Check into issue with RTPush possibly not handling certain types of errors. Example is when the document retriever can’t find the files, batch completion never occurs.

Appendix A: Code Fragment for Documentum APIs

The following code fragment demonstrates the Documentum DFC calls which will likely be required for processing package updates in the FDsys Publish server.

Note that the code below has not been tested. Its purpose is to simply illustrate the Documentum calls required.

```
ProcessUpdates(String DQLStatement) {

  //*** Setup Documentum Client and Session ***

  DfClientX clientx = new DFClientX();
  IDfClient client = clientx.getLocalClient();
  IDfSessionManager sessionMgr = client.newSessionManager():

  IDfLoginInfo loginInfo = clientx.getLoginInfo();
  loginInfo.setUser(userName);
  loginInfo.setPassword(password);

  sessionMgr.setIdentity(IDfSessionManager.ALL_DOCBASES, loginInfo);

  IDfSession session =
        sessionMgr.getSession("Documentum Doc Base Name")


  //*** Query Dctm for packages that need to be published ***

  IDfQuery q = new DfQuery();
  query.setDQL(DQLStatement);

  IDfCollection col = query.execute(session, DfQuery.DF_READ_QUERY);

  while (col.next()) {
    // *** Use col.getString("attr-name"),
    //     col.getId("attr-name"), and
    //     col.getInt("attr-name") to fetch package metadata

    // if is_published = NO or
    //    publish_change_date != first_publish_date
      // also removes empty parent directories:
      removePackageFromACPCache(packageId);
      removePackageFromACPSearchEngine(packageId);

    // if is_published = YES
      addPackage(packageIdVal, packageDctmId);

  }

  col.close()
  sessionMgr.release(session);

}



// *** Process the Folder ***

addPackage(String packageId, IDfId dctmId) {
  String packagePath =
         /* Create the path based on the MD5 of the package ID */

  processFolder(packagePath, dctmId);
  addPackageToSearchEngine(packageId);
}


processFolder(String packagePath, IDfId parentFolderId) {
  IDfFolder folder = (IDfFolder) session.getObject(parentFolderId);

  // if the folder is not public
  //    (use doc.getString("attr-name") as appropriate)
  //    then RETURN without exporting

  IDfCollection folderList = folder.getContents(null);

  while (folderList.next()) {
    IDfTypedObject obj = folderList.getTypedObject();

    // *** if obj.getString("r_object_type") says it's a document

      IDfId docId = obj.getString("r_object_id");
      processDocument(packagePath + "/" +
                obj.getString("object_name"), packageId, docId);

    // *** if obj.getString("r_object_type") says it's a folder

      IDfId childFolderId = obj.getString("r_object_id");
      processFolder(packagePath + "/" +
                  obj.getString("object_name") , childFolderId );
  }
  folderList.close()
}



processDocument(String path, String packageId, String docId) {
  IDfExportOperation eo = clientx.getExportOperation();

  IDfDocument doc =
            (IDfDocument) mySession.getObject(new DfId(docId));
  // if the document is not public
  //    (use doc.getString("attr-name") as appropriate)
  //    then RETURN without exporting

  IDfExportNode node = (IDfExportNode)eo.add(doc);
  node.setFilePath(path + "/" + doc.getObjectName());
  eo.execute()
}
```

Appendix B: fdsys.xml Example

The following is an example of the fdsys.xml file, which is produced by the FDsys parsers and is used as the input for creating the FASTXML file for indexing granules in FAST.

Note that the following file is incomplete. It is missing many metadata values produced during the content submission process.

```
<?xml version="1.0"?>
<FDsys doctype="FR" from="parser">
  <packageid>fr01no06</packageid>
  <category>Regulatory Information</category>
  <collection>Federal Register</collection>
  <branch>Executive</branch>
  <government-author-1>National Archives and Records
Administration</government-author-1>
  <government-author-2>Federal Register Office</government-author-2>
  <issuedate>2006-11-01</issuedate>
  <volume>71</volume>
  <issue>211</issue>

 <granules>
  <granule type="node" grantype="section" gid="154" parentgid="0"
order="1">
    <ancestors><g id="0"/></ancestors>
    <title>Rules and Regulations</title>
  </granule>

  <granule type="leaf" grantype="RulesAndRegulations" gid="1"
parentgid="154" order="2">
    <ancestors><g id="0"/><g id="154"/></ancestors>
    <pages from="64111" to="64113"/>
    <file>granules/granule1.txt</file>
    <agency order="1">DEPARTMENT OF TRANSPORTATION</agency>
    <agency order="2">Federal Aviation Administration</agency>
    <agency-list>DEPARTMENT OF TRANSPORTATION, Federal Aviation
Administration</agency-list>
    <effectivedate>2006-10-28</effectivedate>
    <issuedate>2006-11-01</issuedate>
    <title>Reservation System for Unscheduled Arrivals at Chicago's
O'Hare International Airport</title>
    <migrateddocid>fr01no06-1</migrateddocid>
    <docketno>FAA-2005-19411</docketno>
    <rinnumber>RIN 2120-AI47</rinnumber>
    <billingcode>4910-13-P</billingcode>
    <frdocno>06-9000</frdocno>
    <rawaction>Final rule; extension of expiration date.</rawaction>
    <summary>This action extends the expiration date of Special
Federal Aviation Regulation (SFAR) No. 105 through October 31, 2008.
... </summary>
    <rawdate>This final rule is effective on October 28, 2006, and
SFAR No. 105 published at 70 FR 39610 (July 8, 2005), as amended at
70 FR 66255 (November 2, 2005), ...</rawdate>
    <rawcontact>Gerry Shakley, System Operations Services, Air
Traffic Organization; Telephone: (202) 267-9424; E-mail: ...
</rawcontact>
    <cfr title="14">
      <part>93</part>
    </cfr>
  </granule>
.
.
.
```

Appendix C: Sample FASTXML file

The following is a sample FASTXML file which is produced by the index XSL template program. Note that this is only an early sample, and is missing many metadata values which will be required later.

The shaded portion shown below is the "search schema", or the search.xml portion of the file, after it has been wrapped with <![CDATA[ ]]> tags. The search schema is copied directly into the FAST indexes and allows for complex metadata search without having to add all of the search elements to the FAST index profile.

```
<documents>
  <document>
  <dgid>fr01no06-1</dgid>
  <element name="url"><value>
    file:///C:/Documents%20and%20Settings/Paul%20Nelson/Desktop/gpo-
fr/Pkgfr01no06/granules/granule1.txt
  </value></element>
   <element name="language"><value>en</value></element>
   <element name="getpath"><value>
file:///C:/Documents%20and%20Settings/Paul%20Nelson/Desktop/gpo-
fr/Pkgfr01no06/granules/granule1.txt
   </value></element>
   <element name="granuleid"><value>1</value></element>
   <element name="title"><value>
Reservation System for Unscheduled Arrivals at Chicago's O'Hare
International Airport
</value></element>
   <element name="issuedate"><value>2006-11-01</value></element>
   <element name="effectivedate"><value>2006-10-28</value></element>
   <element name="volume"><value>71</value></element>
   <element name="issue"><value>211</value></element>
   <element name="volissue">
     <value>Vol. 71;Vol. 71/Issue 211</value>
   </element>
   <element name="frompage"><value>64111</value></element>
   <element name="topage"><value>64113</value></element>
   <element name="abstract"><value>
This action extends the expiration date of Special Federal Aviation
Regulation (SFAR) No. 105 through October 31, 2008. This action is
necessary to maintain the reservation system established for
unscheduled arrivals at O'Hare International Airport consistent with
the newly adopted limitations imposed on scheduled operations at the
airport.
   </value></element>
   <element name="rinnumber"><value>RIN 2120-AI47</value></element>
   <element name="docketno"><value>FAA-2005-19411</value></element>
   <element name="section">
     <value>Rules and Regulations</value>
   </element>
   <element name="agencies"><value>
DEPARTMENT OF TRANSPORTATION;Federal Aviation Administration;
   </value></element>
   <element name="cfrtitlepart">
     <value>14 CFR;14 CFR/Part 93;</value>
  </element>
  <element name="xml"><value><![CDATA[
    <fdsys type="FR">
      <packageid>fr01no06</packageid>
      <granuleid>1</granuleid>
      <date type="issue">2006-11-01</date>
      <date type="effective">2006-10-28</date>
      <id type="fr">
        <cite>71 FR 64111</cite>
        <volume>71</volume>
        <issue>211</issue>
        <page>64111</page>
      </id>
      <title>Reservation System for Unscheduled Arrivals ...</title>
      <fields>
        <action>Final rule; extension of expiration date. </action>
        <summary>This action extends the expiration ... </summary>
        <dates>This final rule is effective on October ... </dates>
        <contact>Gerry Shakley, System Operations ... </contact>
      </fields>
      <parents>
        <section level="1">Rules and Regulations</section>
      </parents>
      <org type="branch">
        <name>Executive</name>
        <org><name>DEPARTMENT OF TRANSPORTATION</name></org>
        <org><name>Federal Aviation Administration</name></org>
      </org>
      <refs>
        <id type="cfr">
          <cite>14 CFR Part 93</cite>
          <title>14</title>
          <part>93</part>
        </id>
        <id type="docket"><cite>FAA-2005-19411</cite></id>
        <id type="rin"><cite>RIN 2120-AI47</cite></id>
        <id type="billing"><cite>4910-13-P</cite></id>
      </refs>
    </fdsys> ]]>
  </value></element>
</document>
```

Appendix D: Sample "index.xsl"

The purpose of "index.xsl" is to transform the metadata from the fdsys.xml file into FAST index profile fields (the FASTXML format from Appendix C).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output indent="yes"/>
<xsl:key name="granid" match="//granule" use="@gid"/>
<xsl:preserve-space elements="value"/>
<xsl:template match="/">
  <documents>
    <xsl:for-each select="/FDsys/granules/granule[@type='leaf']">
      <document>
        <dgid><xsl:value-of select="/FDsys/packageid"/>-<xsl:value-
of select="@gid"/></dgid>

        <element name="url">
<value>file:///C:/Documents%20and%20Settings/Paul%20Nelson/Desktop/g
po-fr/Pkgfr01no06/<xsl:value-of select="file"/></value>
        </element>

        <element name="language"><value>en</value></element>

        <element name="getpath">
<value>file:///C:/Documents%20and%20Settings/Paul%20Nelson/Desktop/g
po-fr/Pkgfr01no06/<xsl:value-of select="file"/></value>
        </element>

        <element name="granuleid">
          <value><xsl:value-of select="@gid"/></value>
        </element>

        <element name="title">
          <value><xsl:value-of select="title"/></value>
        </element>

        <element name="issuedate">
          <value><xsl:value-of select="issuedate"/></value>
        </element>

        <xsl:if test="effectivedate">
          <element name="effectivedate">
            <value><xsl:value-of select="effectivedate"/></value>
          </element>
        </xsl:if>

        <element name="volume">
          <value><xsl:value-of select="/FDsys/volume"/></value>
        </element>
        <element name="issue">
          <value><xsl:value-of select="/FDsys/issue"/></value>
        </element>

        <element name="volissue">
          <value>Vol. <xsl:value-of select="/FDsys/volume"/>;Vol.
<xsl:value-of select="/FDsys/volume"/>/Issue <xsl:value-of
select="/FDsys/issue"/></value>
        </element>

        <element name="frompage">
          <value><xsl:value-of select="pages/@from"/></value>
        </element>

        <element name="topage">
          <value><xsl:value-of select="pages/@to"/></value>
        </element>

        <xsl:if test="summary">
          <element name="abstract">
            <value><xsl:value-of select="summary"/></value>
          </element>
        </xsl:if>

        <xsl:if test="rinnumber">
          <element name="rinnumber">
            <value><xsl:value-of select="rinnumber"/></value>
          </element>
        </xsl:if>

        <xsl:if test="docketno">
          <element name="docketno">
            <value><xsl:value-of select="docketno"/></value>
          </element>
        </xsl:if>

        <xsl:call-template name="insertparent">
          <xsl:with-param name="getvalue" select="'title'"/>
          <xsl:with-param name="gid" select="@parentgid"/>
          <xsl:with-param name="grantype" select="'part'"/>
          <xsl:with-param name="elementname" select="'part'"/>
        </xsl:call-template>

        <xsl:call-template name="insertparent">
          <xsl:with-param name="getvalue" select="'title'"/>
          <xsl:with-param name="gid" select="@parentgid"/>
          <xsl:with-param name="grantype" select="'section'"/>
          <xsl:with-param name="elementname" select="'section'"/>
        </xsl:call-template>
        <xsl:if test="agency">
          <element name="agencies">
            <value><xsl:for-each select="agency"><xsl:value-of
select="."/>;</xsl:for-each></value>
          </element>
        </xsl:if>

        <!-- NOTE:  Following code assumes CFRs are already sorted
by @title -->
        <xsl:if test="cfr">
          <element name="cfrtitlepart">
            <value><xsl:for-each select="cfr"><xsl:value-of
select="@title"/> CFR;<xsl:for-each select="part"><xsl:value-of
select="../@title"/> CFR/Part <xsl:value-of select="."/>;</xsl:for-
each></xsl:for-each></value>
          </element>
        </xsl:if>

        <element name="xml">
          <value>
            <xsl:value-of select="'&lt;![CDATA['"  disable-output-
escaping="yes"/>
            <xsl:call-template name="searchxml">
              <xsl:with-param name="gid" select="@gid"/>
            </xsl:call-template>
            <xsl:value-of select="']]&gt;'"  disable-output-
escaping="yes"/>
          </value>
        </element>

      </document>
    </xsl:for-each>
  </documents>
</xsl:template>

<xsl:template name="insertparent">
  <xsl:param name="getvalue"/>
  <xsl:param name="gid"/>
  <xsl:param name="grantype"/>
  <xsl:param name="elementname"/>
  <xsl:for-each select="key('granid',$gid)">
    <xsl:choose>
      <xsl:when test="@grantype=$grantype">
        <element><xsl:attribute name="name"><xsl:value-of
select="$elementname"/></xsl:attribute>
          <value><xsl:value-of
select="child::*[name()=$getvalue]"/></value>
        </element>
      </xsl:when>
      <xsl:when test="@parentgid!='0'">
        <xsl:call-template name="insertparent">
```

```
          <xsl:with-param name="getvalue" select="$getvalue"/>
          <xsl:with-param name="gid" select="@parentgid"/>
          <xsl:with-param name="grantype" select="$grantype"/>
          <xsl:with-param name="elementname" select="$elementname"/>
        </xsl:call-template>
      </xsl:when>
    </xsl:choose>
  </xsl:for-each>
</xsl:template>

<xsl:template name="padnumber"><xsl:param
name="ival"/><xsl:choose><xsl:when test="$ival&lt;10"><xsl:value-of
select="concat('  ',string($ival))"/></xsl:when><xsl:when
test="$ival&lt;100"><xsl:value-of select="concat('
',string($ival))"/></xsl:when><xsl:otherwise><xsl:value-of
select="$ival"/></xsl:otherwise></xsl:choose></xsl:template>

<xsl:import href="mods.xsl"/>

</xsl:stylesheet>
```

Appendix E: Sample "mods.xsl" file

The "mods.xsl" file produces a search XML file for each granule to be search. This schema is designed for easy and intuitive searching. See the "The Search Schema" document for more details.

Note that the "mods.xsl" file is imported into the "index.xsl" file, and its contents are included into the FAST "xml" scope search field. You can see an example of its output in the shaded portion of the FASTXML file in Appendix C.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output indent="yes"/>
<xsl:key name="granid" match="//granule" use="@gid"/>
<xsl:preserve-space elements="value"/>
<xsl:param name="gid"/>

<xsl:template match="/">
  <xsl:call-template name="searchxml">
    <xsl:with-param name="gid" select="$gid"/>
  </xsl:call-template>
</xsl:template>

<xsl:template name="searchxml">
  <xsl:param name="gid"/>
  <fdsys>
    <xsl:attribute name="type"><xsl:value-of
select="/FDsys/@doctype"/></xsl:attribute>
    <xsl:for-each select="/FDsys/granules/granule[@type='leaf' and
@gid=$gid]">
      <xsl:copy-of select="/FDsys/packageid"/>
      <granuleid><xsl:value-of select="@gid"/></granuleid>
      <date type="issue"><xsl:value-of select="issuedate"/></date>
      <xsl:if test="effectivedate"><date
type="effective"><xsl:value-of
select="effectivedate"/></date></xsl:if>
      <id type="fr">
        <cite><xsl:value-of select="/FDsys/volume"/> FR <xsl:value-
of select="pages/@from"/></cite>
        <xsl:copy-of select="/FDsys/volume"/>
        <xsl:copy-of select="/FDsys/issue"/>
        <page><xsl:value-of select="pages/@from"/></page>
      </id>
      <xsl:copy-of select="title"/>

      <fields>
        <xsl:if test="rawaction"><action><xsl:value-of
select="rawaction"/></action></xsl:if>
```

```
        <xsl:if test="summary"><summary><xsl:value-of
select="summary"/></summary></xsl:if>
        <xsl:if test="rawdate"><dates><xsl:value-of
select="rawdate"/></dates></xsl:if>
        <xsl:if test="rawcontact"><contact><xsl:value-of
select="rawcontact"/></contact></xsl:if>
      </fields>

      <parents>
        <xsl:call-template name="insertparents">
          <xsl:with-param name="gid" select="@parentgid"/>
          <xsl:with-param name="level" select="1"/>
        </xsl:call-template>
      </parents>

      <org type="branch">
        <name>Executive</name>
        <xsl:for-each select="agency">
          <org><name><xsl:value-of select="."/></name></org>
        </xsl:for-each>
      </org>

      <refs>
        <xsl:for-each select="cfr">
          <xsl:for-each select="part">
            <id type="cfr">
              <cite><xsl:value-of select="../@title"/> CFR Part
<xsl:value-of select="."/></cite>
              <title><xsl:value-of
select="../@title"/></title><part><xsl:value-of select="."/></part>
            </id>
          </xsl:for-each>
        </xsl:for-each>
        <xsl:if test="docketno"><id type="docket"><cite><xsl:value-
of select="docketno"/></cite></id></xsl:if>
        <xsl:if test="rinnumber"><id type="rin"><cite><xsl:value-of
select="rinnumber"/></cite></id></xsl:if>
        <xsl:if test="billingcode"><id
type="billing"><cite><xsl:value-of
select="billingcode"/></cite></id></xsl:if>
        <xsl:if test="presdoctype">
          <id type="presdoc">
            <xsl:attribute name="type">Presidential <xsl:value-of
select="presdoctype"/></xsl:attribute>
            <cite>Presidential <xsl:value-of select="presdoctype"/>
<xsl:if test="presno"><xsl:value-of select="' '"/><xsl:value-of
select="presno"/></xsl:if> of <xsl:value-of
select="presdate"/></cite>
            <xsl:if test="presno"><number><xsl:value-of
select="presno"/></number></xsl:if>
```

```
            <xsl:if test="presdate"><date type="presdoc"><xsl:value-
of select="presdate"/></date></xsl:if>
          </id>
        </xsl:if>
      </refs>

    </xsl:for-each>
  </fdsys>
</xsl:template>

<xsl:template name="insertparents">
  <xsl:param name="gid"/>
  <xsl:param name="level"/>
  <xsl:for-each select="key('granid',$gid)">
    <xsl:element name="{string(@grantype)}">
      <xsl:attribute name="level"><xsl:value-of select="$level"/>
      </xsl:attribute><xsl:value-of select="title"/>
    </xsl:element>
    <xsl:if test="@parentgid!='0'">
      <xsl:call-template name="insertparents">
        <xsl:with-param name="gid" select="@parentgid"/>
        <xsl:with-param name="level" select="$level+1"/>
      </xsl:call-template>
    </xsl:if>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```