Extending DDF

Table of Contents

1. License

2. Overview

2.1. Building DDF

Follow these procedures to build DDF from source code.

2.1.1. Prerequisites

- Install J2SE 8 SDK.
- Verify that the `JAVA_HOME` environment variable is set to the newly installed JDK location, and that the PATH includes `%JAVA_HOME%\bin` (for Windows) or `$JAVA_HOME$/bin` (*nix).
- Install Git, if not previously installed.
- Install Maven 3.1.0 or later. Verify that the `PATH` includes the `MVN_HOME/bin` directory.
    - In addition, access to a Maven repository with the latest project artifacts and dependencies is necessary in order for a successful build. The following sample `settings.xml` (the default settings file) can be used to access the public repositories with the required artifacts. For more help on how to use the `settings.xml` file, refer to the Maven settings reference page.

Sample `settings.xml` file

```
<settings>
<!-- If proxy is needed
<proxies>
<proxy>
</proxy>
</proxies>
-->
</settings>
```

TIP

Handy Tip on Encrypting Passwords

See this Maven guide on how to encrypt the passwords in your `settings.xml`.

2.1.2. Procedures

Clone the DDF Repository

Using HTTPS

```
https://github.com/codice/ddf.git
```

Using SSH

```
git@github.com:codice/ddf.git
```

NOTE	Generally, SSH is faster than HTTPS, but requires setting an SSH git and passphrase with Github. Additionally, there may be restrictions on the use of SSH on some networks.

Run the Build

- Build command example for one individual repository.

```
# Build is run from the top level of the specified repository in a command line prompt or
terminal.
cd ddf-support
mvn clean install

# At the end of the build, a BUILD SUCCESS will be displayed.
```

NOTE	The zip distribution of DDF is contained in the DDF app in the distribution/ddf/target directory after the DDF app is built.
NOTE	It may take several moments for Maven to download the required dependencies in the first build. Build times may vary based on network speed and machine specifications.
WARNING	In certain circumstances, the build may fail due to a `java.lang.OutOfMemory: Java heap space` error. This error is due to the large number of sub-modules in the DDF build, which causes the heap space to run out in the main Maven JVM. To fix this issue, set the system variable `MAVEN_OPTS` with the value `-Xmx2048m` before running the build. Example on Linux system with the bash shell: `export MAVEN_OPTS='-Xmx2048m'`

2.1.3. Troubleshooting Build Errors on ddf-admin and ddf-ui on a Windows Platform

Currently, the developers are using the following tools:

Name	Version
bower	1.3.2
node.js	v0.10.26
npm	1.4.3

NOTE

There have been intermittent build issues during the bower install. The error code that shows is an EPERM related to either 'renaming' files or 'unlinking' files. This issue has been tracked multiple times on the bower github page. The following link contains the most recent issue that was tracked: https://github.com/bower/bower/issues/991

This issue will be closely monitored for a full resolution. Until a proper solution is found, there are some options that may solve the issue.

1. Re-run the build. Occasionally, the issue occurs on first run and will resolve itself on the next.
2. Clean out the cache. There may be a memory issue, and a cache clean may help solve the issue.

```
bower cache clean
npm cache clean
```

   An occasional reinstall may solve the issue.

```
npm uninstall -g bower && npm install -g bower
```

3. Download and use Cygwin to perform the build. This may allow a user to simulate a run on a *nix system, which may not experience these issues.

These options are taken from suggestions provided on github issue tickets. There have been several tickets created and closed, and several workarounds have been suggested. However, it appears that the issue still exists. Once more information develops on the resolution of this issue, this page will be updated.

2.2. Developing for DDF

This section discusses the several extension points and components permitted by the DDF APIs. Using code examples, diagrams, and references to specific instances of each API, this guide provides details on how to develop and extend various DDF components.

2.3. DDF Development Guidelines

NOTE Development requires full knowledge of the DDF Catalog.

DDF is written in Java and requires a moderate amount of experience with the Java programming language, along with Java terminology, such as packages, methods, classes, and interfaces. DDF uses a small OSGi runtime to deploy components and applications. Before developing for DDF, it is necessary that developers have general knowledge on OSGi and the concepts used within. This includes, but is not limited to, Catalog Commands and the following topics:

- The Service Registry
  - How services are registered
  - How to retrieve service references
- Bundles
  - Their role in OSGi
  - How they are developed

Documentation on OSGi can be viewed at the OSGi Alliance website. Helpful literature for beginners includes *OSGi and Apache Felix 3.0 Beginner's Guide* by Walid Joseph Gédéon and *OSGi in Action: Creating Modular Applications in Java* by Richard Hall, Karl Pauls, Stuart McCulloch, and David Savage.

2.3.1. Recommended Hardware

Because of its modular nature, DDF may require a few or many system resources, depending on which bundles and features are deployed. In general, DDF will take advantage of available memory and processors. A 64-bit JVM is required, and a typical installation is performed on a single machine with 16GB of memory and eight processor cores.

The DDF source code is not tied to any particular IDE. However, if a developer is interested in setting up the Eclipse IDE, they can view the Sonatype guide on developing with Eclipse.

2.3.2. Getting Set Up

To develop on DDF, access to the source code via Github is required.

Integrated Development Environments (IDE)

The DDF source code is not tied to any particular IDE. However, if a developer is interested in setting up the Eclipse IDE, they can view the Sonatype guide on developing with Eclipse.

2.3.3. Formatting Source Code

A code formatter for the Eclipse IDE that can be used across all DDF projects will allow developers to format code similarly and minimize merge issues in the future.

DDF uses an updated version of the Apache ServiceMix Code Formatter for code formatting.

Load the Code Formatter Into the Eclipse IDE

1. Download the Eclipse Code Formatter.
2. In Eclipse, select **Window** **Preferences**. The Preferences window opens.
3. Select **Java** **Code Style** **Formatter**.
4. Select the **Edit...** button and then Select the downloaded file.
5. Select the **OK** button.

Load the Code Formatter Into IntelliJ IDEA

IntelliJ IDEA 13 is capable of importing Eclipse's Code Formatter directly from within IntelliJ without the use of any plugins.

1. Download the IntelliJ Code Formatter.
2. Open **IntelliJ** IDEA.
3. Select **File** **Settings** **Code Style** **Java**.
4. Select **Manage**.
5. Select the **Import** button and select the file.

Format Your Source Code Using Eclipse

A developer may write code and format it before saving.

1. Before the file is saved, highlight all of the source code in the IDE editor window.
2. Right-click on the highlighted code.
3. Select **Source** **Format**. The code formatter is applied to the source code and the file can be saved.

Set Up Save Actions in Eclipse

A developer can also set up Save Actions to format the source code automatically.

1. Open Eclipse.

2. Select **Window** **Preferences** (**Eclipse** **Preferences** on Mac). The Preferences window opens.
3. Select **Java** **Editor** **Save Actions**.
4. Select **Perform the selected actions on save**.
5. Select **Format source code**.
6. Select **Format all lines** or **Format edited lines**, as necessary.
7. Optionally, select **Organize imports** (recommended).
8. Select the **Apply** button.
9. Select the **OK** button.

Format Source Code Using IntelliJ

In the toolbar, select **Code** **Reformat Code** or use the keyboard shortcut `Ctrl-Alt-L`.

2.3.4. DDF Software Versioning

DDF follows the Semantic Versioning White Paper for bundle versioning.

2.3.5. Ensuring Compatibility

Compatibility Goals

The DDF framework, like all software, will mature over time. Changes will be made to improve efficiency, add features, and fix bugs. To ensure that components built for DDF and its sub-frameworks are compatible, developers must use caution when establishing dependencies from developed components.

Guidelines for Maintaining Compatibility

DDF Framework

For components written at the DDF Framework level (see Developing at the Framework Level), adhere to the following specifications:

Standard/Specification	Version	Current Implementation (subject to change)
OSGi Framework	4.2	Apache Karaf 2.x Eclipse Equinox
OSGi Enterprise Specification	4.2	Apache Aries (Blueprint) Apache Felix `FileInstall` and `ConfigurationAdmin`

IMPORTANT	Avoid developing dependencies on the implementations directly, as compatibility in future releases is not guaranteed.

2.3.6. DDF Catalog API

For components written for the DDF Catalog (see Developing Catalog Components), only dependencies on the current major version of the Catalog API should be used. Detailed documentation of the Catalog API can be found in the Catalog API Javadocs.

Dependency	Version Interval	Notes
DDF Catalog API	[2.0, 3.0)	Major version will be incremented (to 3.0) if/when compatibility is broken with the 2.x API.

2.3.7. OGC Filter

An OGC Filter is a Open Geospatial Consortium (OGC) standard that describes a query expression in terms of XML and Key-Value Pairs (KVP).

DDF originally had a custom query representation that some found difficult to understand and implement. In switching to a well-known standard like the OGC Filter, developers benefit from various third party products and third party documentation, as well as any previous experience with the standard. The OGC Filter is used to represent a query to be sent to sources and the Catalog Provider, as well as to represent a Subscription. The OGC Filter provides support for expression processing, such as adding or dividing expressions in a query, but that is not the intended use for DDF.

OGC filter in the DDF Catalog

The DDF Catalog Framework uses the implementation provided by Geotools, which provides a Java representation of the standard.

Geotools originally provided standard Java classes for the OGC Filter Encoding 1.0, under the package name `org.opengis.filter`, which is where `org.opengis.filter.Filter` is located. Java developers should use the Java objects exclusively to complete query tasks, rather than parsing or viewing the XML representation.

2.4. Development Recommendations

2.4.1. Javascript

Avoid using `console.log`

2.4.2. Package Names

Use singular package names.

2.4.3. Author Tags

Author tags are discouraged from being placed in the source code, as they can be a barrier to collaboration and have potential legal ramifications.

2.4.4. Unit Testing

All code should contain unit tests that are able to test out any localized functionality within that class. When working with OSGi, code may have references to various services and other areas that are not available at compile-time. One way to work around the issue of these external dependencies is to use a mocking framework.

NOTE	*Recommended Framework* The recommended framework to use with DDF is Mockito. This test-level dependency is managed by the ddf pom and is used to standardize the version being used across DDF.

2.4.5. Logging

There are many logging frameworks available for Java.

NOTE	*Recommended Framework* To maintain the best compatibility, the recommended logging framework is Simple Logging Facade for Java (SLF4J) (http://www.slf4j.org/), specifically the slf4j-api. SLF4J allows a very robust logging API while letting the backend implementation be switched out seamlessly. Additionally, it is compatible with pax logging and natively implemented by logback.

DDF code uses the first five SLF4J log levels:

1. trace (the least serious)
2. debug
3. info
4. warn
5. error (the most serious)

Examples:

```
//Check if trace is enabled before executing expense XML processing
if (LOGGER.isTraceEnabled()) {
      LOGGER.trace("XML returned: {}", XMLUtils.toString(xml));
}
//It is not necessary to wrap with LOGGER.isTraceEnabled() since slf4j will not construct
the String unless
//trace level is enabled
LOGGER.trace("Executing search: {}", search);
```

2.5. "White Box" DDF Architecture

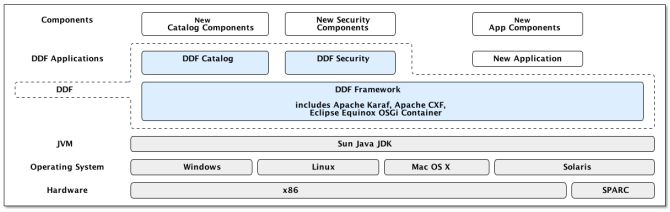


Figure 1. Architecture Diagram

As depicted in the architectural diagram above, DDF runs on top of an OSGi framework, a Java virtual machine (JVM), several choices of operating systems, and the physical hardware infrastructure. The items within the dotted line represent the DDF out-of-the-box.

DDF is a customized and branded distribution of Apache Karaf. DDF could also be considered to be a more lightweight OSGi distribution, as compared to Apache ServiceMix, FUSE ESB, or Talend ESB, all of which are also built upon Apache Karaf. Similar to its peers, DDF incorporates additional upstream dependencies (https://tools.codice.org/#DDFArchitecture-AdditionalUpstreamDependencies).

DDF as a framework hosts DDF applications, which are extensible by adding components via OSGi. The best example of this is the DDF Catalog (API), which offers extensibility via several types of Catalog Components. The DDF Catalog API serves as the foundation for several applications and resides in the applications tier.

The Catalog Components consist of Endpoints, Plugins, Catalog Frameworks, Sources, and Catalog Providers. Customized components can be added to DDF.

2.5.1. Nomenclature

Capability

A general term used to refer to an ability of the system

Application

One or more features that together form a cohesive collection of capabilities

Component

Represents a portion of an Application that can be extended

Bundle

Java Archives (JARs) with special OSGi manifest entries.

Feature

One or more bundles that form an installable unit; Defined by Apache Karaf but portable to other OSGi containers.

2.5.2. OSGi Core

DDF makes use of OSGi v4.2 to provide several capabilities:

- Has a Microkernel-based foundation, which is lightweight due to its origin in embedded systems.
- Enables integrators to easily customize components to run on their system.
- Software applications are deployed as OSGi components, or bundles. Bundles are modules that can be deployed into the OSGi container (Eclipse Equinox OSGi Framework by default).
- Bundles provide flexibility allowing integrators to choose the bundles that meet their mission needs.
- Bundles provide reusable modules that can be dropped in any container.
- Provides modularity, module-based security, and low-level services, such as Hypertext Transfer Protocol (HTTP), logging, events (basic publish/subscribe), and dependency injection.
- Implements a dynamic component model that allows application updates without downtime. Components can be added or updated in a running system.
- Standardized Application Configuration (ConfigurationAdmin and MetaType)

OSGi is not an acronym, but if more context is desired the name Open Specifications Group Initiative has been suggested.

More information on OSGi is available at http://www.osgi.org/.

2.5.3. Built on Apache Karaf

Apache Karaf is a FOSS product that includes an OSGi framework and adds extra functionality, including:

Admin Console

Useful for configuring applications, installing/uninstalling features, and viewing services such as metrics.

Command Console

Provides command line administration of the OSGi container. All functionality in the Command Console can also be performed via this command line console.

Logging

Provides centralized logging to `data/log/ddf.log`. Security logging is provided at `data/log/security.log`. Ingest error logging can be viewed in `data/log/ingest_error.log`.

Provisioning

Of libraries or applications.

Security

Provides a security framework based on Java Authentication and Authorization Service (JAAS).

Deployer

Provides hot deployment of new bundles by dropping them into the `<INSTALL_DIR>/deploy` directory.

Blueprint

Provides an implementation of the OSGi Blueprint Container specification that defines a dependency injection framework for dealing with dynamic configuration of OSGi services.

- DDF uses the Apache Aries implementation of Blueprint. More information can be found at Blueprint.

Spring DM

An alternative dependency injection framework. DDF is not dependent on specific dependency injection framework, but Blueprint is recommended.

2.5.4. Additional Upstream Dependencies

DDF is a customized distribution of Apache Karaf, and therefore includes all the capabilities of Apache Karaf. DDF also includes additional FOSS components to provide a richer set of capabilities. Integrated components include their own dependencies, but at the platform level, DDF includes the following upstream dependencies:

Apache CXF

Apache CXF is an open source services framework. CXF helps build and develop services using front end programming APIs, such as JAX-WS and JAX-RS. More information can be found at http://cxf.apache.org.

Apache Commons

Provides a set of reusable Java components that extends functionality beyond that provided by the standard JDK (More info available at http://commons.apache.org)

OSGeo GeoTools

Provides spatial object model and fundamental geometric functions, which are used by DDF spatial criteria searches. More information can be found at http://geotools.org/.

Joda Time

Provides an enhanced, easier to use version of Java date and time classes. More information can be found at http://joda-time.sourceforge.net.

For a full list of dependencies, refer to the Software Version Description Document (SVDD).

2.6. OSGi Services

Services consist of:

An API Bundle

The API, written as Java interfaces, defines the contract of the service and should, to the extent possible, reference only those concrete classes that are loaded by the root classloader. These classes being in the `java.*` packages. These exceptions to the `java.*` rule can be made:

- Extra interfaces can be declared by the API and used as input parameters and return values from API methods.
- Because of their complexity and relative permanence, generated JAXB classes can be exported from an API bundle for use by its consumers.

At least one implementation bundle

As the intent of loosely coupled services is to allow a variety of implementations to be deployed into the container, it is common for there to be more than one concrete implementation of a service. However, that is not a requirement. A single implementation can suffice. It should include a `blueprint.xml` associating the implementation class(es) with interface(s), providing any other wiring of beans, services, and metadata necessary, and registering with the container.

2.6.1. Dependency Injection Frameworks

DDF uses resource injection to retrieve and register services to the OSGi registry. There are many resource injection frameworks that are used to complete these operations. Blueprint and Spring DM are both used by DDF.

NOTE | It is recommended to use Blueprint over Spring DM wherever possible.

There are many tutorials and guides available on the Internet for both of these frameworks.

2.6.2. Blueprint - Retrieving a Service Instance

Blueprint example of retrieving and injecting services

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

 <reference id="ddfCatalogFramework" interface="ddf.catalog.CatalogFramework" />

 <bean class="my.sample.NiftyEndpoint" >
    <argument ref="ddfCatalogFramework" />
 </bean>

</blueprint>
```

Line #	Action
3	Retrieves a Service from the Registry
6	Instantiates a new object, injecting the retrieved Service as a constructor argument

2.6.3. Spring DM - Retrieving a Service Instance

Spring DM example of retrieving and injecting services

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:osgi="http://www.springframework.org/schema/osgi">

  <osgi:reference id="ddfCatalogFramework" interface="ddf.catalog.CatalogFramework" />

  <bean class="my.sample.NiftyEndpoint">
     <constructor-arg ref="ddfCatalogFramework" />
  </bean>
</beans>
```

Line #	Action
5	Retrieves a Service from the Registry
8	Instantiates a new object, injecting the retrieved Service as a constructor argument

2.6.4. Blueprint - Registering a Service into the Registry

Creating a bean and registering it into the Service Registry

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="transformer" class="my.sample.NiftyTransformer"/>

  <service ref="transformer" interface="ddf.catalog.transform.QueryResponseTransformer"
/>

</blueprint>
```

Line #	Action
3	Instantiates a new object
5	Registers the object instance created in Line 3 as a service that implements the `ddf.catalog.transform.QueryResponseTransformer` interface

2.6.5. Packaging Capabilities as Bundles

Services and code are physically deployed to DDF using bundles. The bundles within DDF are created using the maven bundle plug-in. Bundles are Java JAR files that have additional metadata in the `MANIFEST.MF` that is relevant to an OSGi container.

The best resource for learning about the structure and headers in the manifest definition is in section 3.6 of the OSGi Core Specification. The bundles within DDF are created using the maven bundle plug-in, which uses the BND tool. Bundles are Java JAR files that have additional metadata in the `MANIFEST.MF` file that is relevant to an OSGi container.

TIP

Alternative Bundle Creation Methods

Using Maven is not necessary to create bundles. Many alternative tools exist, and OSGi manifest files can also be created by hand, although hand-editing should be avoided by most developers.

Creating a Bundle

Bundle Development Recommendations

Avoid creating bundles by hand or editing a manifest file

Many tools exist for creating bundles, notably the Maven Bundle plugin, which handle the details of OSGi configuration and automate the bundling process including generation of the manifest file.

Always make a distinction on which imported packages are `optional` or `required`

Requiring every package when not necessary can cause an unnecessary dependency ripple effect among bundles.

Embedding is an implementation detail

Using the `Embed-Dependency` instruction provided by the `maven-bundle-plugin` will insert the specified jar(s) into the target archive and add them to the `Bundle-ClassPath`. These jars and their contained packages/classes are not for public consumption; they are for the internal implementation of this service implementation only.

Bundles should never be embedded

Bundles expose service implementations; they do not provide arbitrary classes to be used by other bundles.

Bundles should expose service implementations

This is the corollary to the previous rule. Bundles should not be created when arbitrary concrete classes are being extracted to a library. In that case, a library/jar is the appropriate module packaging type.

Bundles should generally only export service packages

If there are packages internal to a bundle that comprise its implementation but not its public manifestation of the API, they should be excluded from export and kept as private packages.

Concrete objects that are not loaded by the root classloader should not be passed in or out of a bundle

This is a general rule with some exceptions (JAXB generated classes being the most prominent example). Where complex objects need to be passed in or out of a service method, an interface should be defined in the API bundle.

Bundles separate contract from implementation and allow for modularized development and deployment of functionality. For that to be effective, they must be defined and used correctly so inadvertent coupling does not occur. Good bundle definition and usage leads to a more flexible environment.

Maven Bundle Plugin

Below is a code snippet from a Maven `pom.xml` for creating an OSGi Bundle using the Maven Bundle plugin.

Maven `pom.xml`

```
...
<packaging>bundle</packaging>
...
<build>
...
  <plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
    <configuration>
      <instructions>
        <Bundle-Name>DDF DOCS</Bundle-Name>
        <Export-Package />
        <Bundle-SymbolicName>ddf.docs</Bundle-SymbolicName>
        <Import-Package>
          ddf.catalog,
          ddf.catalog.*
        </Import-Package>
      </instructions>
    </configuration>
  </plugin>
...
</build>
...
```

Third Party and Utility Bundles

It is recommended to avoid building directly on included third party and utility bundles. These components do provide utility and reuse potential; however, they may be upgraded or even replaced at anytime as bug fixes and new capabilities dictate. For example, Web services may be built using CXF. However, the distributions frequently upgrade CXF between releases to take advantage of new features. If building on these components, be aware of the version upgrades with each distribution release.

Instead, component developers should package and deliver their own dependencies to ensure future compatibility. For example, if re-using a bundle, the specific bundle version that you are depending on should be included in your packaged release, and the proper versions should be referenced in your bundle(s).

Deploying a Bundle

A bundle is typically installed in one of two ways:

1. Installed as a feature
2. Hot deployed in the `/deploy` directory

The fastest way to deploy a created bundle during development is to copy it to the `/deploy` directory of a running DDF. This directory checks for new bundles and deploys them immediately. According to Karaf documentation, "Karaf supports hot deployment of OSGi bundles by monitoring JAR files inside the `[home]/deploy` directory. Each time a JAR is copied in this folder, it will be installed inside the runtime. It can be updated or deleted and changes will be handled automatically. In addition, Karaf also supports exploded bundles and custom deployers (Blueprint and Spring DM are included by default)." Once deployed, the bundle should come up in the Active state, if all of the dependencies were properly met. When this occurs, the service is available to be used.

Verifying Bundle State

To verify if a bundle is deployed and running, go to the running command console and view the status.

- Execute the `list` command.
- If the name of the bundle is known, the `list` command can be piped to the `grep` command to quickly find the bundle.

The example below shows how to verify if a Client is deployed and running.

Verifying with grep

```
ddf@local>list | grep -i example
[ 162] [Active     ] [       ] [   ] [ 80] DDF :: Registry :: example Client (2.0.0)
```

The state is `Active`, indicating that the bundle is ready for program execution.

2.6.6. Additional Bundling Resources

- Blueprint
  - http://aries.apache.org/modules/blueprint.html
  - http://www.ibm.com/developerworks/opensource/library/os-osgiblueprint/
  - http://static.springsource.org/osgi/docs/2.0.0.M1/reference/html/blueprint.html
- Spring DM
  - http://www.springsource.org/osgi
  - Lessons Learned from it-agile (PDF)
  - http://www.martinlippert.org/events/OOP2010-OSGiLessonsLearned.pdf
- Creating Bundles
  - http://blog.springsource.com/2008/02/18/creating-osgi-bundles/

- Bundle States
  - http://static.springsource.org/osgi/docs/1.2.1/reference/html/bnd-app-ctx.html

2.6.7. Working with Features

Features XML files group other features and/or bundle(s) for ease of installation/uninstallation.

```
<feature name="catalog-app" install="auto" version="2.9.1"
    description="The DDF Catalog provides a framework for storing, searching, processing,
and transforming information.\nClients typically perform query, create, read, update, and
delete (QCRUD) operations against the Catalog.\nAt the core of the Catalog functionality
is the Catalog Framework, which routes all requests and responses through the system,
invoking additional processing per the system configuration.::DDF Catalog">
    <feature>platform-app</feature>
    <feature>catalog-core</feature>
    <feature>catalog-core-metricsplugin</feature>
    <feature>catalog-core-sourcemetricsplugin</feature>
    <feature>catalog-transformer-thumbnail</feature>
    <feature>catalog-transformer-metadata</feature>
    <feature>catalog-transformer-xsltengine</feature>
    <feature>catalog-transformer-resource</feature>
    <feature>catalog-rest-endpoint</feature>
    <feature>catalog-opensearch-endpoint</feature>
    <feature>catalog-opensearch-source</feature>
    <feature>catalog-transformer-json</feature>
    <feature>catalog-transformer-atom</feature>
    <feature>catalog-transformer-geoformatter</feature>
    <feature>catalog-transformer-xml</feature>
    <feature>catalog-transformer-tika</feature>
    <feature>catalog-security-plugin</feature>
    <feature>catalog-admin-module-sources</feature>
    <feature>catalog-core-backupplugin</feature>
    <feature>catalog-plugin-jpeg2000</feature>
</feature>

<feature name="catalog-core" install="manual" version="2.9.1"
    description="Catalog Core feature containing the API, third party bundles necessary
to run ddf-core.">
    <feature>catalog-core-api</feature>
    <bundle>mvn:ddf.catalog.core/catalog-core-commons/2.9.1</bundle>
    <bundle>mvn:ddf.catalog.core/catalog-core-camelcomponent/2.9.1</bundle>
    <bundle>mvn:ddf.measure/measure-api/2.9.1</bundle>
    <bundle>mvn:org.codice.thirdparty/picocontainer/1.2_1</bundle>
    <bundle>mvn:org.codice.thirdparty/vecmath/1.3.2_1</bundle> <!-- for GeoTools -->
    <bundle>mvn:org.codice.thirdparty/geotools-suite/8.4_2</bundle>
    <bundle>mvn:org.codice.thirdparty/jts/1.12_1</bundle>
    <bundle>mvn:ddf.catalog.core/catalog-core-federationstrategy/2.9.1</bundle>
    <bundle>mvn:org.codice.thirdparty/lucene-core/3.0.2_1</bundle>
    <bundle>mvn:ddf.catalog.core/ddf-pubsub/2.9.1</bundle>
    <bundle>mvn:ddf.catalog.core/catalog-core-eventcommands/2.9.1</bundle>
    <bundle>mvn:ddf.catalog.core/ddf-pubsub-tracker/2.9.1</bundle>
    <bundle>mvn:ddf.catalog.core/catalog-core-urlresourcereader/2.9.1</bundle>
    <bundle>mvn:ddf.catalog.core/filter-proxy/2.9.1</bundle>
    <bundle>mvn:ddf.catalog.core/catalog-core-commands/2.9.1</bundle>
```

```
    <bundle>mvn:ddf.catalog.core/catalog-core-metacardgroomerplugin/2.9.1</bundle>
    <bundle>mvn:ddf.catalog.core/metacard-type-registry/2.9.1</bundle>
    <bundle>mvn:ddf.catalog.core/catalog-core-standardframework/2.9.1</bundle>
    <bundle>mvn:ddf.catalog.core/catalog-core-resourcesizeplugin/2.9.1</bundle>

    <configfile finalname="/data/solr/metacard_cache/conf/solrconfig.xml"
>mvn:ddf.platform.solr/platform-solr-server-standalone/2.9.1/xml/solrconfig</configfile>
    <configfile finalname="/data/solr/metacard_cache/conf/schema.xml"
>mvn:ddf.platform.solr/platform-solr-server-standalone/2.9.1/xml/schema</configfile>
    <configfile finalname="/data/solr/metacard_cache/conf/protwords.txt"
>mvn:ddf.platform.solr/platform-solr-server-standalone/2.9.1/txt/protwords</configfile>
    <configfile finalname="/data/solr/metacard_cache/conf/stopwords_en.txt"
>mvn:ddf.platform.solr/platform-solr-server-
standalone/2.9.1/txt/stopwords_en</configfile>
    <configfile finalname="/data/solr/metacard_cache/conf/stopwords.txt"
>mvn:ddf.platform.solr/platform-solr-server-standalone/2.9.1/txt/stopwords</configfile>
    <configfile finalname="/data/solr/metacard_cache/conf/synonyms.txt"
>mvn:ddf.platform.solr/platform-solr-server-standalone/2.9.1/txt/synonyms</configfile>
</feature>
```

2.6.8. Making Sure a Features File Will Display Properly in the Installer

In order to ensure that the installer can correctly interpret and display application details, there are several guidelines that should be followed when creating the features file for the application.

- Be sure that only one feature in the `features.xml` has the `auto-install` tag.

  ```
  install='auto'
  ```

  This is the feature that the installer displays to the user (name, description, version, etc.). It is typically named after the application itself and the description provides a complete application description.

- Be sure that the one feature specified to `auto-install` has a complete list of all of its dependencies in order to ensure the dependency tree can be constructed correctly.

Auto-starting an Application Feature

Within the `features.xml` file for an application, one feature will have the install attribute set to `auto`. Within this feature, refer to any dependencies of the application as well as any features that should start automatically. Other features should have install set to `manual`.

The following example demonstrates configuring features to be auto-started. The naming convention for this feature is typically “application name” + "`-app`," as shown.

Auto-start features

```
<feature name="catalog-app" install="auto">
    <feature>platform-app</feature>
    <feature>catalog-core</feature>
    <feature>catalog-core-metricsplugin</feature>
    <feature>catalog-core-sourcemetricsplugin</feature>
    <feature>catalog-transformer-thumbnail</feature>
</feature>
```

2.7. Developing DDF Applications

The DDF applications are comprised of components, packaged as Karaf features, which are collections of OSGi bundles. These features can be installed/uninstalled using the Admin Console or Command Console. DDF applications also consist of one or more OSGi bundles and, possibly, supplemental external files. These applications are packaged as Karaf KAR files for easy download and installation. These applications can be stored on a file system or a Maven repository.

A KAR file is a Karaf-specific archive format (*K*araf *AR*chive). It is a jar file that contains a feature descriptor file and one or more OSGi bundle jar files. The feature descriptor file identifies the application's name, the set of bundles that need to be installed, and any dependencies on other features that may need to be installed.

2.7.1. Describing Application Services

Given the modular nature of OSGi, some applications perform operations on the services themselves. In order to present, identify, and manipulate the services, they need descriptive identifying information. Any service that implements the `Describable` interface in `org.codice.ddf.platform.services.common` will have an obligation to provide this information. The relevant fields are as follows:

- ID: a unique identifier for the service
- Title: the informal name for the service
- Description: a short, human-consumable description of the service
- Organization: the name of the organization that wrote the service
- Version: the current version of the service (example: 1.0)

The only field with stringent requirements is the ID field. Format should be `[product].[component]` such as `ddf.metacards` or `ddf.platform`; while the [**component**] within a [**product**] may simply be a module or bundle name, the [**product**] itself should be the unique name of the plug-in or integration that belongs to the organization provided. Note that `ddf` as a [**product**] is reserved for core features only and is not meant to be used during extension or integration.

2.7.2. Creating a KAR File

The recommended method for creating a KAR file is to use the `features-maven-plugin`, which has a `create-kar` goal. This goal reads all of the features specified in the feature's descriptor file. For each feature in this file, it resolves the bundles defined in the feature. All bundles are then packaged into the KAR archive.

create-kar Goal Example

```
<plugin>
<groupId>org.apache.karaf.tooling</groupId>
<artifactId>features-maven-plugin</artifactId>
<version>2.2.5</version>
<executions>
<execution>
<id>create-kar</id>
<goals>
<goal>create-kar</goal>
</goals>
<configuration>
<descriptors>
<!-- Add any other <descriptor> that the features file may reference here -->
</descriptors>
<!--
Workaround to prevent the target/classes/features.xml file from being included in the
kar file since features.xml already included in kar's repository directory tree.
Otherwise, features.xml would appear twice in the kar file, hence installing the
same feature twice.
Refer to Karaf forum posting at http://karaf.922171.n3.nabble.com/Duplicate-feature-
repository-entry-using-archive-kar-to-build-deployable-applications-td3650850.html
-->
<resourcesDir>/Users/phillip/Documents/workspace/release/ddf/distribution/docs/target/doe
sNotExist</resourcesDir>

<!--
Location of the features.xml file. If it references properties that need to be filtered,
e.g., 2.9.1, it will need to be
filtered by the maven-resources-plugin.
-->
<featuresFile>/Users/phillip/Documents/workspace/release/ddf/distribution/docs/target/cla
sses/features.xml</featuresFile>

<!-- Name of the kar file (.kar extension added by default). If not specified, defaults
to docs-2.9.1 -->
<finalName>ddf-ifis-2.9.1</finalName>
</configuration>
</execution>
    </executions>
</plugin>
```

Examples of how KAR files are created for DDF components can be found in the DDF source code under the ddf/distribution/ddf-kars directory.

The `.kar` file generated should be deployed to the application author's maven repository. The URL to

the application's KAR file in this Maven repository should be the installation URL that is used.

2.7.3. Including Data Files in a KAR File

The developer may need to include data or configuration file(s) in a KAR file. An example of this is a properties file for the JDBC connection properties of a catalog provider.

It is recommended that:

- Any `data/configuration` files be placed under the `src/main/resources` directory of the maven project. Sub-directories under `src/main/resources` can be used, e.g., `etc/security`.
- The Maven project's pom file should be updated to attach each `data/configuration` file as an artifact (using the `build-helper-maven-plugin`).
- Add each `data/configuration` file to the KAR file using the `<configfile>` tag in the KAR's `features.xml` file.

2.7.4. Installing a KAR File

When the user downloads an application by clicking on the **Installation** link, the application's KAR file is downloaded. To install manually, the KAR file can be placed in the `<DDF_INSTALL_DIR>/deploy` directory of the running DDF instance. DDF then detects that a file with a `.kar` file extension has been placed in this monitored directory, unzips the KAR file into the `<DDF_INSTALL_DIR>/system` directory, and installs the bundle(s) listed in the KAR file's feature descriptor file. To install via the Admin Console: . Navigate to https://localhost:8993/admin . Click the **Manage** button in the upper right . Click the **Add an Application** tile . Upload the KAR file via the popup window . Click **Save Changes** to activate The new application can be viewed via the Admin Console's Active Applications list.

Developing Application Configuration Modules

An application within DDF is a collection of bundles contained in a KAR file that may or may not have configurations associated with it. Plugins are used to advertise applications. These configuration module plugins are often used to add user interface elements to make the use of the DDF simpler and/or more intuitive.

Creating an Application Configuration Module

This example demonstrates a plugin that allows the DDF to use the Admin UI.

1. Create an application plugin to advertise your configuration by extending `AbstractApplicationPlugin`.

```
import org.codice.ddf.admin.application.plugin.AbstractApplicationPlugin;

public class SourcesPlugin extends AbstractApplicationPlugin {
    /**
     * Constructor.
     */

    public SourcesPlugin() {
        this.displayName = "Sources";
        this.iframeLocation = URI.create("/admin/sources/index.html");
        List<String> apps = new ArrayList<String>();
        apps.add("catalog-app");
        this.setAssociations(apps);
    }
}
```

2. Configure as shown with a name, URI, and any dependency applications.

3. Register the application with Blueprint through a `blueprint.xml` file.

   `blueprint.xml`

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="
  http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

    <bean id="appModule" class="org.codice.ui.admin.applications.ApplicationModule"
></bean>

    <service interface="org.codice.ddf.ui.admin.api.module.AdminModule" ref="appModule"
/>

</blueprint>
```

4. Create application to use this configuration.

Including KAR Files

Sometimes a developer may need to include data or configuration file(s) in a KAR file. An example of this would be a properties file for the JDBC connection properties of a catalog provider.

It is recommended that:

- Any data/configuration files be placed under the `src/main/resources` directory of the maven project.

(Sub-directories under `src/main/resources` can also be used, e.g., `etc/security`)

- The maven project's pom file should be updated to attach each data/configuration file as an artifact (using the `build-helper-maven-plugin`)
- Add each data/configuration file to the KAR file by using the `<configfile>` tag in the KAR's `features.xml` file

2.8. Migration API

DDF currently has an experimental API for making bundles migratable. Interfaces in `platform/migration/platform-migratable-api` are used by the system to identify bundles that provide implementations for import (*coming soon*) and export operations.

NOTE	This code is experimental. While this interface is functional and tested, it may change or be removed in a future version of the library.

An export operation can be performed through the Command Console or through the Admin Console. When a export operation is processed, the migration API will do a look-up for all registered OSGi services that are implementing `Migratable` and call their `export()` method.

The services that implement one of the migratable interfaces will be called one at a time, in any order, and do not need to be thread safe. A bundle or a feature can have as many services implementing the interfaces as needed.

2.8.1. `Migratable`

The contract for a `migratable` is stored here. It is used by both sub-interfaces `ConfigurationMigratable` and `DataMigratable`.

WARNING	Do not implement `Migratable` directly; it is intended for use only as a common base interface. Instead, the appropriate sub-interface should be used.

2.8.2. `ConfigurationMigratable`

This is the base interface that must be implemented by all bundles or features that need to export and/or import system related settings (e.g. bundle specific Java properties, XML or JSON configuration files) during system migration. The information exported must allow the new system to have the same configuration as the original system. Only bundle or feature specific settings need be handled. All configurations stored in OSGi's `ConfigurationAdmin` will automatically be migrated and do not need to be managed by implementors of this class.

Also, any other data not related to the system's configuration and settings (e.g., data stored in Solr) should be handled by a different service that implements the `DataMigratable` interface, not by implementors of this class.

2.8.3. `DataMigratable`

This is the interface that must be implemented by all bundles or features that need to export and/or import data during system migration. The data is not mandatory for the system to come up (e.g., data stored in Solr) and doesn't affect the system's configuration, i.e., without this data, the new system will still have the same configuration as the original one.

Any system related configuration and settings should be handled by a different service that implements the `ConfigurationMigratable` interface, not by implementors of this class.

2.9. Developing CometD Clients for Asynchronous Search and Retrieval

DDF uses the CometD framework to perform asynchronous operations on the catalog. This is most apparent in the UI application, which is able to perform multiple queries and display them asynchronously as the results are returned. CometD has a comprehensive manual that details how to interact with and best use the framework with a variety of clients (e.g., javascript, java, perl, and python).

The SearchUI code can be used as reference implementation of how a JavaScript client can be created to integrate with the CometD endpoint. Examples are included below for convenience.

2.9.1. General Operations

Initialization

CometD offers both Dojo and jQuery bindings that allow CometD to be easily used with those frameworks. For more information on the individual bindings, refer to the JavaScript Library page in the CometD reference manual. The UI application uses the jQuery library. To initialize the CometD connection, an instance of CometD must be created and configured to point to the server and call the handshake, which creates the network connection. The CometD endpoint is exposed at `/search/cometd`, and it is recommended to not use websockets when connecting.

Example Initialization with JQuery

```
// create a reference to the standard cometd object
Cometd.Comet = $.cometd;

// disable websocket protocol
Cometd.Comet.websocketEnabled = false;

// configure the location of the cometd endpoint on the server
Cometd.Comet.configure({
url: 'http://server:8181/search/cometd'
});

// create network connection to server
Cometd.Comet.handshake({});
```

Subscribe

A client can asynchronously receive information from the server using subscriptions. Once the client subscribes to a particular topic, it is sent information when the server sends a response on that topic. Subscriptions are performed in the UI to retrieve information for notifications, tasks, and query results.

Example Subscription

```
// subscribe to a topic, calling the function whenever a new message comes in
var subscription = Cometd.Comet.subscribe("/ddf/notifications/**",
function(resp) { ... } );

// unsubscribe from a topic
Cometd.Comet.unsubscribe(subscription);
```

NOTE Further documentation is available from CometD/javascript_subscribe.html.

Publish

Publishing allows clients to send information to the server. This allows the clients to perform queries on the server and perform operations on tasks, such as canceling a download.

Example Publish

```
// perform a query on the server where data contains the search criteria
Cometd.Comet.publish('/service/query', { data: { ... } });
```

Publish Notifications

Any application running in DDF can publish notifications that can be viewed by the Search UI or received by another notifications client.

1. Set a properties map containing entries for each of the parameters listed above in the Usage section.
2. Set the OSGi event topic to `ddf/notification/<application-name>/<notification-type>` Notice that there is no preceding slash on an OSGi event topic name, while there is one on the CometD channel name. The OSGi event topic corresponds to the CometD channel this is published on.
3. Post the notification to the OSGi event defined in the previous step.

Example for Publishing Notification

```
Dictionary <String, Object> properties = new Hashtable<String,
Object>();
properties.put("application", "Downloads");
properties.put("title", resourceResponse.getResource().getName());
Long sysTimeMillis = System.currentTimeMillis();
properties.put("message", generateMessage(status,
resourceResponse.getResource().getName(), bytes, sysTimeMillis,
detail));
properties.put("user", getProperty(resourceResponse, USER));
properties.put("status", "Completed");
properties.put("bytes", 1024);
properties.put("timestamp", sysTimeMillis);
Event event = new Event("ddf/notification/catalog/downloads",
properties);
eventAdmin.postEvent(event);
```

NOTE	Futher documentation is available at http://docs.cometd.org/2/reference/javascript_publish.html.

Retrieving Persisted Notifications and Activities

To retrieve persisted notifications or activities, publish an empty message on the corresponding base channel. This will trigger a response to an awaiting Cometd subscription. Refer to Developing CometD Clients for Asynchronous Search and Retrieval for instructions on how to publish to a CometD channel.

Example: Persistence Retrieval

```
Cometd.Comet.publish("/ddf/notifications", {});
Cometd.Comet.publish("/ddf/activities", {});
```

NOTE

Asynchronous Query

An asynchronous query is performed using the CometD Endpoint, which is currently packaged with the DDF Search UI application.

2.10. Web Service Security Architecture

The Web Service Security (WSS) functionality that comes with DDF is integrated throughout the system. This is a central resource describing how all of the pieces work together and where they are located within the system.

DDF comes with a **Security Framework** and **Security Services**. The Security Framework is the set of APIs that define the integration with the DDF framework and the Security Services are the reference implementations of those APIs built for a realistic end-to-end use case.

2.10.1. Security Framework

The DDF Security Framework utilizes Apache Shiro as the underlying security framework. The classes mentioned in this section will have their full package name listed, to make it easy to tell which classes come with the core Shiro framework and which are added by DDF.

Subject

`ddf.security.Subject <extends> org.apache.shiro.subject.Subject`

The Subject is the key object in the security framework. Most of the workflow and implementations revolve around creating and using a Subject. The Subject object in DDF is a class that encapsulates all information about the user performing the current operation. The Subject can also be used to perform permission checks to see if the calling user has acceptable permission to perform a certain action (e.g., calling a service or returning a metacard). This class was made DDF-specific because the Shiro interface cannot be added to the Query Request property map.

Table 1. Implementations of Subject:

Classname	Description
ddf.security.impl.SubjectImpl	Extends `org.apache.shiro.subject.support.DelegatingSubject`

Security Manager

`ddf.security.service.SecurityManager`

The Security Manager is a service that handles the creation of Subject objects. A proxy to this service should be obtained by an endpoint to create a Subject and add it to the outgoing `QueryRequest`. The Shiro framework relies on creating the subject by obtaining it from the current thread. Due to the multi-threaded and stateless nature of the DDF framework, utilizing the Security Manager interface makes retrieving Subjects easier and safer.

Table 2. Implementations of Security Managers:

Classname	Description
`ddf.security.service.SecurityManagerImpl`	This implementation of the Security Manager handles taking in both `org.apache.shiro.authc`. AuthenticationToken and `org.apache.cxf.ws.security.tokenstore.SecurityToken` objects.

Authentication Tokens

`org.apache.shiro.authc.AuthenticationToken`

Authentication Tokens are used to verify authentication of a user when creating a subject. A common use-case is when a user is logging directly in to the DDF framework.

Classname	Description
`ddf.security.service.impl.cas.CasAuthenticationToken`	This Authentication Token is used for authenticating a user that has logged in with CAS. It takes in a proxy ticket which can be validated on the CAS server.

2.10.2. Realms

Authenticating Realms

`org.apache.shiro.realm.AuthenticatingRealm`

Authenticating Realms are used to authenticate an incoming authentication token and create a Subject on successfully authentication.

Implementations of Authenticating Realms that come with DDF:

Classname	Description
`ddf.security.realm.sts.StsRealm`	This realm delegates authentication to the Secure Token Service (STS). It creates a `RequestSecurityToken` message from the incoming Authentication Token and converts a successful STS response into a Subject.

Authorizing Realms

`org.apache.shiro.realm.AuthorizingRealm`

Authorizing Realms are used to perform authorization on the current Subject. These are used when performing both Service AuthZ and Filtering. They are passed in the `AuthorizationInfo` of the Subject along with the Permissions of the object wanting to be accessed. The response from these realms is a true (if the Subject has permission to access) or false (if the Subject does not).

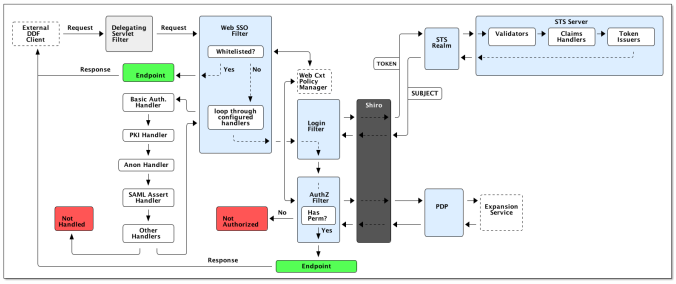
Table 3. Other implementations of the Security API within DDF

Classname	Description
`org.codice.ddf.platform.filter.delegate.DelegateServletFilter`	The `DelegateServletFilter` detects any servlet filters that have been exposed as OSGi services and places them in-order in front of any servlet or web application running on the container.
`org.codice.ddf.security.filter.websso.WebSSOFilter`	This filter serves as the main security filter that works in conjunction with a number of handlers to protect a variety of contexts, each using different authentication schemes and policies.
`org.codice.ddf.security.handler.saml.SAMLAssertionHandler`	This handler is executed by the WebSSOFilter for any contexts configured to use it. This handler should always come first when configured in the Web Context Policy Manager, as it provides a caching capability to web contexts that use it. The handler will first check for the existence of a cookie named "org.codice.websso.saml.token" to extract a Base64 + deflate SAML assertion from the request. If an assertion is found it will be converted to a `SecurityToken`. Failing that, the handler will check for a JSESSIONID cookie that might relate to a current SSO session with the container. If the JSESSIONID is valid, the SecurityToken will be retrieved from the cache in the LoginFilter.
`org.codice.ddf.security.handler.basic.BasicAuthenticationHandler`	Checks for basic authentication credentials in the http request header. If they exist, they are retrieved and passed to the `LoginFilter` for exchange.
`org.codice.ddf.security.handler.pki.PKIHandler`	Handler for PKI based authentication. X509 chain will be extracted from the HTTP request and converted to a `BinarySecurityToken`.
`org.codice.ddf.security.handler.guest.GuestHandler`	Handler that allows guest user access via a guest user account. The guest account credentials are configured via the org.codice.ddf.security.claims.guest.GuestClaimsHandler. The `GuestHandler` also checks for the existence of basic auth credentials or PKI credentials that might be able to override the use of the anonymous user.
`org.codice.ddf.security.filter.login.LoginFilter`	This filter runs immediately after the WebSSOFilter and exchanges any authentication information found in the request with a Subject via Shiro.
`org.codice.ddf.security.filter.authorization.AuthorizationFilter`	This filter runs immediately after the `LoginFilter` and checks any permissions assigned to the web context against the attributes of the user via Shiro.

Classname	Description
`org.apache.shiro.realm.AuthenticatingRealm`	This is an abstract authenticating realm that exchanges an `org.apache.shiro.authc.AuthenticationToken` for a `ddf.security.Subject` in the form of an `org.apache.shiro.authc.AuthenticationInfo`
`ddf.security.realm.sts.StsRealm`	This realm is an implementation of `org.apache.shiro.realm.AuthenticatingRealm` and connects to an STS (configurable) to exchange the authentication token for a Subject.
`ddf.security.service.AbstractAuthorizingRealm`	This is an abstract authorizing realm that takes care of caching and parsing the Subject's `AuthorizingInfo` and should be extended to allow the implementing realm focus on making the decision.
`ddf.security.pdp.realm.AuthZRealm`	This realm performs the authorization decision and may or may not delegate out to the external XACML processing engine. It uses the incoming permissions to create a decision. However, it is possible to extend this realm using the ddf.security.policy.extension.PolicyExtension interface. This interface allows an integrator to add additional policy information to the PDP that can't be covered via its generic matching policies. This approach is often easier to configure for those that are not familiar with XACML. Note that no `PolicyExtension` implementations are provided out of the box.
`org.codice.ddf.security.validator.*`	A number of STS validators are provided for X.509 (BinarySecurityToken), UsernameToken, SAML Assertion, and DDF custom tokens. The DDF custom tokens are all `BinarySecurityTokens` that may have PKI or username/password information as well as an authentication realm (correlates to JAAS realms installed in the container). The authentication realm allows an administrator to restrict which services they wish to use to authenticate users. For example: installing the `security-sts-ldaplogin` feature will enable a JAAS realm with the name "ldap". This realm can then be specified on any context using the Web Context Policy Manager. That realm selection is then passed via the token sent to the STS to determine which validator to use.

WARNING	An update was made to the SAML Assertion Handler to pass SAML assertions via headers instead of cookies. Cookies are still accepted and processed to maintain legacy federation compatibility, but only headers are used when federating out. This means that it is still possible to federate and pass a machine's identity, but federation of a user's identity will ONLY work when federating from 2.7.x to 2.8.x+ or between 2.8.x+ and 2.8.x+.

2.10.3. Securing REST



The delegating servlet filter is topmost filter for all web contexts. It loads in all security filters. The first filter used is the Web SSO filter. It reads from the web context policy manager and functions as the first decision point. If the request is from a whitelisted context, no further authentication is needed and the request goes directly to the desired endpoint. If the context is not on the whitelist, the filter will attempt to get a handler for the context. The filter loops through all configured context handlers until one signals that it has found authentication information that it can use to build a token. This configuration can be changed by modifying the web context policy manager configuration. If unable to resolve the context, the filter will return an authentication error and the process stops. If a handler is successfully found, an auth token is assigned and the request continues to the login filter. The Login Filter receives a token and return a subject. To retrieve the subject, the token is sent through Shiro to the STS Realm where the token will be exchanged for a SAML assertion through a SOAP call to an STS server. If the Subject is returned, the request moves to the Authorization Filter to check permissions on the user. If the user has the correct permissions to access that web context, the request is allowed to hit the endpoint.

NOTE	This diagram does not yet include the SAML 2.0 Web SSO integration.

2.10.4. Encryption Service

The encryption service and encryption command, which are based on Jasypt, provide an easy way for developers to add encryption capabilities to DDF.

Encryption Command

An encrypt security command is provided with DDF that allows plain text to be encrypted. This is useful when displaying password fields in a GUI.

Below is an example of the security:encrypt command used to encrypt the plain text "myPasswordToEncrypt". The output, `bR9mJpDVo8bTRwqGwIFxHJ5yFJzatKwjXjIo/8USWm8=`, is the encrypted

value.

```
ddf@local>security:encrypt myPasswordToEncrypt

bR9mJpDVo8bTRwqGwIFxHJ5yFJzatKwjXjIo/8USWm8=
```

2.10.5. Filtering

Metacard filtering is performed in a Access plugin that occurs after a query has been performed.

How Filtering Works

Each metacard result will contain security attributes that are populated by the CatalogFramework based on the PolicyPlugins (Not provided! You must create your own plugin for your specific metadata!) that populates this attribute. The security attribute is a HashMap containing a set of keys that map to lists of values. The metacard is then processed by a filter plugin that creates a `KeyValueCollectionPermission` from the metacard's security attribute. This permission is then checked against the user subject to determine if the subject has the correct claims to view that metacard. The decision to filter the metacard eventually relies on the PDP (`feature:install security-pdp-authz`). The PDP returns a decision, and the metacard will either be filtered or allowed to pass through.

The security attributes populated on the metacard are completely dependent on the type of the metacard. Each type of metacard must have its own PolicyPlugin that reads the metadata being returned and returns the metacard's security attribute. If the subject permissions are missing during filtering, all resources will be filtered.

Example (represented as simple XML for ease of understanding):

```
<metacard>
    <security>
        <map>
            <entry key="entry1" value="A,B" />
            <entry key="entry2" value="X,Y" />
            <entry key="entry3" value="USA,GBR" />
            <entry key="entry4" value="USA,AUS" />
        </map>
    </security>
</metacard>
```

```
<user>
    <claim name="claim1">
        <value>A</value>
        <value>B</value>
    </claim>
    <claim name="claim2">
        <value>X</value>
        <value>Y</value>
    </claim>
    <claim name="claim3">
        <value>USA</value>
    </claim>
    <claim name="claim4">
        <value>USA</value>
    </claim>
</user>
```

In the above example, the user's claims are represented very simply and are similar to how they would actually appear in a SAML 2 assertion. Each of these user (or subject) claims will be converted to a KeyValuePermission object. These permission objects will be implied against the permission object generated from the metacard record. In this particular case, the metacard might be allowed if the policy is configured appropriately because all of the permissions line up correctly.

2.10.6. Filter a New Type of Metacard

To enable filtering on a new type of record, implement a PolicyPlugin that is able to read the string metadata contained within the metacard record. Note that, in DDF, there is no default plugin that parses a metacard. A plugin must be created to create a policy for the metacard.

2.10.7. Security Token Service

The Security Token Service (STS) is a service running in DDF that generates SAML v2.0 assertions. These assertions are then used to authenticate a client allowing them to issue other requests, such as ingests or queries to DDF services.

The STS is an extension of Apache CXF-STS. It is a SOAP web service that utilizes WS-Trust. The generated SAML assertions contain attributes about a user and is used by the Policy Enforcement Point (PEP) in the secure endpoints. Specific configuration details on the bundles that come with DDF can be found on the Security STS application page. This page details all of the STS components that come out of the box with DDF, along with configuration options, installation help, and which services they import and export.

The STS server contains validators, claim handlers, and token issuers to process incoming requests. When a request is received, the validators first ensure that it is valid. The validators verifies authentication against configured services, such as LDAP, DIAS, PKI. If the request is found to be

invalid, the process ends and an error is returned. Next, the claims handlers determine how to handle the request, adding user attributes or properties as configured. The token issuer creates a SAML 2.0 assertion and associates it with the subject. The STS server sends an assertion back to the requestor, which is used in both SOAP and REST cases.

Using the Security Token Service (STS)

The STS can be used to generate SAML v2.0 assertions via a SOAP web service request. Out of the box, the STS supports authentication from existing SAML tokens, CAS proxy tickets, username/password, and x509 certificates. It also supports retrieving claims using LDAP and properties files.

STS Claims Handlers

Claims handlers are classes that convert the incoming user credentials into a set of attribute claims that will be populated in the SAML assertion. An example in action would be the LDAPClaimsHandler that takes in the user's credentials and retrieves the user's attributes from a backend LDAP server. These attributes are then mapped and added to the SAML assertion being created. Integrators and developers can add more claims handlers that can handle other types of external services that store user attributes.

Add a Custom Claims Handler

Description

A claim is an additional piece of data about a subject that can be included in a token along with basic token data. A claims manager provides hooks for a developer to plug in claims handlers to ensure that the STS includes the specified claims in the issued token.

Motivation

A developer may want to add a custom claims handler to retrieve attributes from an external attribute store.

Steps

The following steps define the procedure for adding a custom claims handler to the STS.

1. The new claims handler must implement the `org.apache.cxf.sts.claims.ClaimsHander` interface.

```
/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements. See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership. The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */

package org.apache.cxf.sts.claims;

import java.net.URI;
import java.util.List;

/**
 * This interface provides a pluggable way to handle Claims.
 */
public interface ClaimsHandler {

    List<URI> getSupportedClaimTypes();

    ClaimCollection retrieveClaimValues(RequestClaimCollection claims, ClaimsParameters
parameters);

}
```

2. Expose the new claims handler as an OSGi service under the `org.apache.cxf.sts.claims.ClaimsHandler` interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

    <bean id="CustomClaimsHandler" class=
"security.sts.claimsHandler.CustomClaimsHandler" />

    <service ref="customClaimsHandler" interface=
"org.apache.cxf.sts.claims.ClaimsHandler"/>

</blueprint>
```

3. Deploy the bundle.

If the new claims handler is hitting an external service that is secured with SSL/TLS, a developer may need to add the root CA of the external site to the DDF trustStore and add a valid certificate into the DDF keyStore. For more information on certificates, refer to [Configuring a Java Keystore for Secure Communications].

STS WS-Trust WSDL Document

NOTE This XML file is found inside of the STS bundle and is named `ws-trust-1.4-service.wsdl`.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:tns="http://docs.oasis-open.org/ws-sx/ws-trust/200512/"
xmlns:wstrust="http://docs.oasis-open.org/ws-sx/ws-trust/200512/" xmlns:wsdl=
"http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsap10="http://www.w3.org/2006/05/addressing/wsdl" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp=
"http://www.w3.org/ns/ws-policy" xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:wsam=
"http://www.w3.org/2007/05/addressing/metadata" targetNamespace="http://docs.oasis-
open.org/ws-sx/ws-trust/200512/">
    <wsdl:types>
        <xs:schema elementFormDefault="qualified" targetNamespace="http://docs.oasis-
open.org/ws-sx/ws-trust/200512">
            <xs:element name="RequestSecurityToken" type=
"wst:AbstractRequestSecurityTokenType"/>
            <xs:element name="RequestSecurityTokenResponse" type=
"wst:AbstractRequestSecurityTokenType"/>
            <xs:complexType name="AbstractRequestSecurityTokenType">
                <xs:sequence>
                    <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="Context" type="xs:anyURI" use="optional"/>
                <xs:anyAttribute namespace="##other" processContents="lax"/>
            </xs:complexType>
            <xs:element name="RequestSecurityTokenCollection" type=
"wst:RequestSecurityTokenCollectionType"/>
            <xs:complexType name="RequestSecurityTokenCollectionType">
                <xs:sequence>
                    <xs:element name="RequestSecurityToken" type=
"wst:AbstractRequestSecurityTokenType" minOccurs="2" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
            <xs:element name="RequestSecurityTokenResponseCollection" type=
"wst:RequestSecurityTokenResponseCollectionType"/>
            <xs:complexType name="RequestSecurityTokenResponseCollectionType">
                <xs:sequence>
                    <xs:element ref="wst:RequestSecurityTokenResponse" minOccurs="1"
maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:anyAttribute namespace="##other" processContents="lax"/>
            </xs:complexType>
        </xs:schema>
    </wsdl:types>
    <!-- WS-Trust defines the following GEDs -->
    <wsdl:message name="RequestSecurityTokenMsg">
        <wsdl:part name="request" element="wst:RequestSecurityToken"/>
```

```
    </wsdl:message>
    <wsdl:message name="RequestSecurityTokenResponseMsg">
        <wsdl:part name="response" element="wst:RequestSecurityTokenResponse"/>
    </wsdl:message>
    <wsdl:message name="RequestSecurityTokenCollectionMsg">
        <wsdl:part name="requestCollection" element="wst:RequestSecurityTokenCollection
"/>
    </wsdl:message>
    <wsdl:message name="RequestSecurityTokenResponseCollectionMsg">
        <wsdl:part name="responseCollection" element=
"wst:RequestSecurityTokenResponseCollection"/>
    </wsdl:message>
    <!-- This portType an example of a Requestor (or other) endpoint that
         Accepts SOAP-based challenges from a Security Token Service -->
    <wsdl:portType name="WSSecurityRequestor">
        <wsdl:operation name="Challenge">
            <wsdl:input message="tns:RequestSecurityTokenResponseMsg"/>
            <wsdl:output message="tns:RequestSecurityTokenResponseMsg"/>
        </wsdl:operation>
    </wsdl:portType>
    <!-- This portType is an example of an STS supporting full protocol -->
    <wsdl:portType name="STS">
        <wsdl:operation name="Cancel">
            <wsdl:input wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Cancel" message="tns:RequestSecurityTokenMsg"/>
            <wsdl:output wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RSTR/CancelFinal" message="tns:RequestSecurityTokenResponseMsg"/>
        </wsdl:operation>
        <wsdl:operation name="Issue">
            <wsdl:input wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Issue" message="tns:RequestSecurityTokenMsg"/>
            <wsdl:output wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RSTRC/IssueFinal" message="tns:RequestSecurityTokenResponseCollectionMsg"/>
        </wsdl:operation>
        <wsdl:operation name="Renew">
            <wsdl:input wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Renew" message="tns:RequestSecurityTokenMsg"/>
            <wsdl:output wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RSTR/RenewFinal" message="tns:RequestSecurityTokenResponseMsg"/>
        </wsdl:operation>
        <wsdl:operation name="Validate">
            <wsdl:input wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Validate" message="tns:RequestSecurityTokenMsg"/>
            <wsdl:output wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RSTR/ValidateFinal" message="tns:RequestSecurityTokenResponseMsg"/>
        </wsdl:operation>
        <wsdl:operation name="KeyExchangeToken">
            <wsdl:input wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
```

```
trust/200512/RST/KET" message="tns:RequestSecurityTokenMsg"/>
            <wsdl:output wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RSTR/KETFinal" message="tns:RequestSecurityTokenResponseMsg"/>
        </wsdl:operation>
        <wsdl:operation name="RequestCollection">
            <wsdl:input message="tns:RequestSecurityTokenCollectionMsg"/>
            <wsdl:output message="tns:RequestSecurityTokenResponseCollectionMsg"/>
        </wsdl:operation>
    </wsdl:portType>
    <!-- This portType is an example of an endpoint that accepts
         Unsolicited RequestSecurityTokenResponse messages -->
    <wsdl:portType name="SecurityTokenResponseService">
        <wsdl:operation name="RequestSecurityTokenResponse">
            <wsdl:input message="tns:RequestSecurityTokenResponseMsg"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="STS_Binding" type="wstrust:STS">
        <wsp:PolicyReference URI="#STS_policy"/>
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="Issue">
            <soap:operation soapAction="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Issue"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="Validate">
            <soap:operation soapAction="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Validate"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="Cancel">
            <soap:operation soapAction="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Cancel"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
```

```
        </wsdl:operation>
        <wsdl:operation name="Renew">
            <soap:operation soapAction="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Renew"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="KeyExchangeToken">
            <soap:operation soapAction="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/KeyExchangeToken"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="RequestCollection">
            <soap:operation soapAction="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/RequestCollection"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsp:Policy wsu:Id="STS_policy">
        <wsp:ExactlyOne>
            <wsp:All>
                <wsap10:UsingAddressing/>
                <wsp:ExactlyOne>
                    <sp:TransportBinding xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702">
                        <wsp:Policy>
                            <sp:TransportToken>
                                <wsp:Policy>
                                    <sp:HttpsToken>
                                        <wsp:Policy/>
                                    </sp:HttpsToken>
                                </wsp:Policy>
                            </sp:TransportToken>
                            <sp:AlgorithmSuite>
```

```
                                <wsp:Policy>
                                    <sp:Basic128/>
                                </wsp:Policy>
                            </sp:AlgorithmSuite>
                            <sp:Layout>
                                <wsp:Policy>
                                    <sp:Lax/>
                                </wsp:Policy>
                            </sp:Layout>
                            <sp:IncludeTimestamp/>
                        </wsp:Policy>
                    </sp:TransportBinding>
                </wsp:ExactlyOne>
                <sp:Wss11 xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702">
                    <wsp:Policy>
                        <sp:MustSupportRefKeyIdentifier/>
                        <sp:MustSupportRefIssuerSerial/>
                        <sp:MustSupportRefThumbprint/>
                        <sp:MustSupportRefEncryptedKey/>
                    </wsp:Policy>
                </sp:Wss11>
                <sp:Trust13 xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702">
                    <wsp:Policy>
                        <sp:MustSupportIssuedTokens/>
                        <sp:RequireClientEntropy/>
                        <sp:RequireServerEntropy/>
                    </wsp:Policy>
                </sp:Trust13>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
    <wsp:Policy wsu:Id="Input_policy">
        <wsp:ExactlyOne>
            <wsp:All>
                <sp:SignedParts xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702">
                    <sp:Body/>
                    <sp:Header Name="To" Namespace="http://www.w3.org/2005/08/addressing
"/>
                    <sp:Header Name="From" Namespace=
"http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="FaultTo" Namespace=
"http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="ReplyTo" Namespace=
"http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="MessageID" Namespace=
```

```
"http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="RelatesTo" Namespace=
"http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="Action" Namespace=
"http://www.w3.org/2005/08/addressing"/>
                </sp:SignedParts>
                <sp:EncryptedParts xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702">
                    <sp:Body/>
                </sp:EncryptedParts>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
    <wsp:Policy wsu:Id="Output_policy">
        <wsp:ExactlyOne>
            <wsp:All>
                <sp:SignedParts xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702">
                    <sp:Body/>
                    <sp:Header Name="To" Namespace="http://www.w3.org/2005/08/addressing
"/>
                    <sp:Header Name="From" Namespace=
"http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="FaultTo" Namespace=
"http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="ReplyTo" Namespace=
"http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="MessageID" Namespace=
"http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="RelatesTo" Namespace=
"http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="Action" Namespace=
"http://www.w3.org/2005/08/addressing"/>
                </sp:SignedParts>
                <sp:EncryptedParts xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702">
                    <sp:Body/>
                </sp:EncryptedParts>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
    <wsdl:service name="SecurityTokenService">
        <wsdl:port name="STS_Port" binding="tns:STS_Binding">
            <soap:address location="http://localhost:8181/services/SecurityTokenService
"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

2.10.8. Example Request and Responses for a SAML Assertion

A client performs a RequestSecurityToken operation against the STS to receive a SAML assertion. The DDF STS offers several different ways to request a SAML assertion. For help in understanding the various request and response formats, samples have been provided. The samples are divided out into different request token types.

Most endpoints that have been used in DDF require the X.509 PublicKey SAML assertion.

2.10.9. BinarySecurityToken (CAS) SAML Security Token Request/Response

BinarySecurityToken (CAS) Sample Request/Response

Request

Sample Request

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <Action xmlns="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/ws-sx/ws-trust/200512/RST/Issue</Action>
        <MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:60652909-faca-
4e4a-a4a7-8a5ce243a7cb</MessageID>
        <To xmlns="http://www.w3.org/2005/08/addressing"
>https://server:8993/services/SecurityTokenService</To>
        <ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
            <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
        </ReplyTo>
        <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd" soap:mustUnderstand="1">
            <wsu:Timestamp wsu:Id="TS-1">
                <wsu:Created>2013-04-29T18:35:10.688Z</wsu:Created>
                <wsu:Expires>2013-04-29T18:40:10.688Z</wsu:Expires>
            </wsu:Timestamp>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
        <wst:RequestSecurityToken xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512">
            <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue</wst:RequestType>
            <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
                <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing">
                    <wsa:Address>https://server:8993/services/SecurityTokenService</wsa:A
ddress>
                </wsa:EndpointReference>
            </wsp:AppliesTo>
            <wst:Claims xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512" Dialect=
"http://schemas.xmlsoap.org/ws/2005/05/identity">
                <ic:ClaimType xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
Optional="true" Uri="
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"/>
                <ic:ClaimType xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
Optional="true" Uri="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
"/>
                <ic:ClaimType xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
Optional="true" Uri="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname"/>
                <ic:ClaimType xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
Optional="true" Uri="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname"/>
                <ic:ClaimType xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
Optional="true" Uri="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"/>
```

```
            </wst:Claims>
            <wst:OnBehalfOf>
                <BinarySecurityToken ValueType="#CAS" EncodingType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary" ns1:Id="
CAS" xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd" xmlns:ns1="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
>U1QtMTQtYUtmcDYxcFRtS0FxZG1pVDMzOWMtY2FzfGh0dHBzOi8vdG9rZW5pc3N1ZXI6ODk5My9zZXJ2aWNlcy9T
ZWN1cml0eVRva2VuU2VydmljZQ==</BinarySecurityToken>
            </wst:OnBehalfOf>
            <wst:TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV2.0</wst:TokenType>
            <wst:KeyType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/PublicKey</wst:KeyType>
            <wst:UseKey>
                <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                    <ds:X509Data>
                        <ds:X509Certificate>
MIIC5DCCAk2gAwIBAgIJAKj7ROPHjo1yMA0GCSqGSIb3DQEBCwUAMIGKMQswCQYDVQQGEwJVUzEQ
MA4GA1UECAwHQXJpem9uYTERMA8GA1UEBwwIR29vZHllYXIxGDAWBgNVBAoMD0xvY2toZWVkIE1h
cnRpbjENMAsGA1UECwwESTRDRTEPMA0GA1UEAwwGY2xpZW50MRwwGgYJKoZIhvcNAQkBFg1pNGNl
QGxtY28uY29tMB4XDTEyMDYyMDE5NDMwOVoXDTIyMDYxODE5NDMwOVowgYoxCzAJBgNVBAYTAlVT
MRAwDgYDVQQIDAdBcml6b25hMREwDwYDVQQHDAhHb29keWVhcjEYMBYGA1UECgwPTG9ja2hlZWQg
TWFydGluMQ0wCwYDVQQLDARJNENFMQ8wDQYDVQQDDAZjbGllbnQxHDAaBgkqhkiG9w0BCQEWDWk0
Y2VAbG1jby5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAIpHxCBLYE7xfDLcITS9SsPG
4Q04Z6S32/+TriGsRgpGTj/7GuMG7oJ98m6Ws5cTYl7nyunyHTkZuP7rBzy4esDIHheyx18EgdSJ
vvACgGVCnEmHndkf9bWUlAOfNaxW+vZwljUkRUVdkhPbPdPwOcMdKg/SsLSNjZfsQIjoWd4rAgMB
AAGjUDBOMB0GA1UdDgQWBBQx11VLtYXLvFGpFdHnhlNW9+lxBDAfBgNVHSMEGDAWgBQx11VLtYXL
vFGpFdHnhlNW9+lxBDAMBgNVHRMEBTADAQH/MA0GCSqGSIb3DQEBCwUAA4GBAHYs2OI0K6yVXzyS
sKcv2fmfw6XCICGTnyA7BOdAjYoqq6wD+33dHJUCFDqye7AWdcivuc7RWJt9jnlfJZKIm2BHcDTR
Hhk6CvjJ14Gf40WQdeMHoX8U8b0diq7Iy5Ravx+zRg7SdiyJUqFYjRh/O5tywXRT1+freI3bwAN0
L6tQ
</ds:X509Certificate>
                    </ds:X509Data>
                </ds:KeyInfo>
            </wst:UseKey>
            <wst:Renewing/>
        </wst:RequestSecurityToken>
    </soap:Body>
</soap:Envelope>
```

Response

Sample Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <Action xmlns="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/ws-sx/ws-trust/200512/RSTRC/IssueFinal</Action>
        <MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:7a6fde04-9013-
41ef-b08b-0689ffa9c93e</MessageID>
        <To xmlns="http://www.w3.org/2005/08/addressing"
>http://www.w3.org/2005/08/addressing/anonymous</To>
        <RelatesTo xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:60652909-faca-
4e4a-a4a7-8a5ce243a7cb</RelatesTo>
        <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd" soap:mustUnderstand="1">
            <wsu:Timestamp wsu:Id="TS-2">
                <wsu:Created>2013-04-29T18:35:11.459Z</wsu:Created>
                <wsu:Expires>2013-04-29T18:40:11.459Z</wsu:Expires>
            </wsu:Timestamp>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
        <RequestSecurityTokenResponseCollection xmlns="http://docs.oasis-open.org/ws-
sx/ws-trust/200512" xmlns:ns2="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" xmlns:ns3="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd" xmlns:ns4="http://www.w3.org/2005/08/addressing"
xmlns:ns5="http://docs.oasis-open.org/ws-sx/ws-trust/200802">
            <RequestSecurityTokenResponse>
                <TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV2.0</TokenType>
                <RequestedSecurityToken>
                    <saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" ID="_BDC44EB8593F47D1B213672605113671" IssueInstant="2013-04-29T18:35:11.370Z"
Version="2.0" xsi:type="saml2:AssertionType">
                        <saml2:Issuer>tokenissuer</saml2:Issuer>
                        <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                            <ds:SignedInfo>
                                <ds:CanonicalizationMethod Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                <ds:SignatureMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                                <ds:Reference URI="#_BDC44EB8593F47D1B213672605113671">
                                    <ds:Transforms>
                                        <ds:Transform Algorithm=
"http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                                        <ds:Transform Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#">
```

```
                                        <ec:InclusiveNamespaces xmlns:ec=
"http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="xs"/>
                                    </ds:Transform>
                                </ds:Transforms>
                                <ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
                                <ds:DigestValue>6wnWbft6Pz5XOF5Q9AG59gcGwLY=</ds:Dige
stValue>
                            </ds:Reference>
                        </ds:SignedInfo>
                        <ds:SignatureValue>h+NvkgXGdQtca3/eKebhAKgG38tHp3i2n5uLLy8xXX
Ig02qyKgEP0FCowp2LiYlsQU9YjKfSwCUbH3WR6jhbAv9zj29CE+ePfEny7MeXvgNl3wId+vcHqti/DGGhhgtO2Mb
x/tyX1BhHQUwKRlcHajxHeecwmvV7D85NMdV48tI=</ds:SignatureValue>
                        <ds:KeyInfo>
                            <ds:X509Data>
                                <ds:X509Certificate>MIIDmjCCAwOgAwIBAgIBBDANBgkqhkiG9
w0BAQQFADB1MQswCQYDVQQGEwJVUzEQMA4GA1UECBMH
QXJpem9uYTERMA8GA1UEBxMIR29vZHllYXIxEDAOBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0V4
YW1wbGUxEDAOBgNVBAsTB0V4YW1wbGUxCzAJBgNVBAMTAkNBMB4XDTEzMDQwOTE4MzcxMVoXDTIz
MDQwNzE4MzcxMVowgaYxCzAJBgNVBAYTAlVTMRAwDgYDVQQIEwdBcml6b25hMREwDwYDVQQHEwhH
b29keWVhcjEQMA4GA1UEChMHRXhhbXBsZTEQMA4GA1UEChMHRXhhbXBsZTEQMA4GA1UECxMHRXhh
bXBsZTEUMBIGA1UEAxMLdG9rZW5pc3N1ZXIxJjAkBgkqhkiG9w0BCQEWF3Rva2VuaXNzdWVyQGV4
YW1wbGUuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDDfktpA8Lrp9rTfRibKdgtxtN9
uB44diiIqq3JOzDGfDhGLu6mjpuHO1hrKItv42hBOhhmH7lS9ipiaQCIpVfgIG63MB7fa5dBrfGF
G69vFrU1Lfi7IvsVVsNrtAEQljOMmw9sxS3SUsRQX+bD8jq7Uj1hpoF7DdqpV8Kb0COOGwIDAQAB
o4IBBjCCAQIwCQYDVR0TBAIwADAsBglghkgBhvhCAQ0EHxYdT3BlblNTTCBHZW5lcmF0ZWQgQ2Vy
dGlmaWNhdGUwHQYDVR0OBBYEFD1mHviop2Tc4HaNu8yPXR6GqWP1MIGnBgNVHSMEgZ8wgZyAFBcn
en6/j05DzaVwORwrteKc7TZOoXmkdzB1MQswCQYDVQQGEwJVUzEQMA4GA1UECBMHQXJpem9uYTER
MA8GA1UEBxMIR29vZHllYXIxEDAOBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0V4YW1wbGUxEDAO
BgNVBAsTB0V4YW1wbGUxCzAJBgNVBAMTAkNBggkAwXk7OcwO7gwwDQYJKoZIhvcNAQEEBQADgYEA
PiTX5kYXwdhmijutSkrObKpRbQkvkkzcyZlO6VrAxRQ+eFeN6NyuyhgYy5K6l/sIWdaGou5iJOQx
2pQYWx1v8Klyl0W22IfEAXYv/epiO89hpdACryuDJpioXI/X8TAwvRwLKL21Dk3k2b+eyCgA0O++
HM0dPfiQLQ99ElWkv/0=</ds:X509Certificate>
                            </ds:X509Data>
                        </ds:KeyInfo>
                    </ds:Signature>
                    <saml2:Subject>
                        <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified" NameQualifier="http://cxf.apache.org/sts">srogers</saml2:NameID>
                        <saml2:SubjectConfirmation Method=
"urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
                            <saml2:SubjectConfirmationData xsi:type=
"saml2:KeyInfoConfirmationDataType">
                                <ds:KeyInfo xmlns:ds=
"http://www.w3.org/2000/09/xmldsig#">
                                    <ds:X509Data>
                                        <ds:X509Certificate>MIIC5DCCAk2gAwIBAgIJAKj7R
OPHjo1yMA0GCSqGSIb3DQEBCwUAMIGKMQswCQYDVQQGEwJVUzEQ
```

```
MA4GA1UECAwHQXJpem9uYTERMA8GA1UEBwwIR29vZHllYXIxGDAWBgNVBAoMD0xvY2toZWVkIE1h
cnRpbjENMAsGA1UECwwESTRDRTEPMA0GA1UEAwwGY2xpZW50MRwwGgYJKoZIhvcNAQkBFg1pNGNl
QGxtY28uY29tMB4XDTEyMDYyMDE5NDMwOVoXDTIyMDYxODE5NDMwOVowgYoxCzAJBgNVBAYTAlVT
MRAwDgYDVQQIDAdBcml6b25hMREwDwYDVQQHDAhHb29keWVhcjEYMBYGA1UECgwPTG9ja2hlZWQg
TWFydGluMQ0wCwYDVQQLDARJNENFMQ8wDQYDVQQDDAZjbGllbnQxHDAaBgkqhkiG9w0BCQEWDWk0
Y2VAbG1jby5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAIpHxCBLYE7xfDLcITS9SsPG
4Q04Z6S32/+TriGsRgpGTj/7GuMG7oJ98m6Ws5cTYl7nyunyHTkZuP7rBzy4esDIHheyx18EgdSJ
vvACgGVCnEmHndkf9bWUlAOfNaxW+vZwljUkRUVdkhPbPdPwOcMdKg/SsLSNjZfsQIjoWd4rAgMB
AAGjUDBOMB0GA1UdDgQWBBQx11VLtYXLvFGpFdHnhlNW9+lxBDAfBgNVHSMEGDAWgBQx11VLtYXL
vFGpFdHnhlNW9+lxBDAMBgNVHRMEBTADAQH/MA0GCSqGSIb3DQEBCwUAA4GBAHYs2OI0K6yVXzyS
sKcv2fmfw6XCICGTnyA7BOdAjYoqq6wD+33dHJUCFDqye7AWdcivuc7RWJt9jnlfJZKIm2BHcDTR
Hhk6CvjJ14Gf40WQdeMHoX8U8b0diq7Iy5Ravx+zRg7SdiyJUqFYjRh/O5tywXRT1+freI3bwAN0
L6tQ</ds:X509Certificate>
                                    </ds:X509Data>
                                </ds:KeyInfo>
                            </saml2:SubjectConfirmationData>
                        </saml2:SubjectConfirmation>
                    </saml2:Subject>
                    <saml2:Conditions NotBefore="2013-04-29T18:35:11.407Z"
NotOnOrAfter="2013-04-29T19:05:11.407Z">
                        <saml2:AudienceRestriction>
                            <saml2:Audience>https://server:8993/services/SecurityToke
nService</saml2:Audience>
                        </saml2:AudienceRestriction>
                    </saml2:Conditions>
                    <saml2:AuthnStatement AuthnInstant="2013-04-29T18:35:11.392Z">
                        <saml2:AuthnContext>
                            <saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:a
c:classes:unspecified</saml2:AuthnContextClassRef>
                        </saml2:AuthnContext>
                    </saml2:AuthnStatement>
                    <saml2:AttributeStatement>
                        <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                            <saml2:AttributeValue xsi:type="xs:string">
srogers</saml2:AttributeValue>
                        </saml2:Attribute>
                        <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                            <saml2:AttributeValue xsi:type="xs:string"
>srogers@example.com</saml2:AttributeValue>
                        </saml2:Attribute>
                        <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                            <saml2:AttributeValue xsi:type="xs:string">
```

```
srogers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">Steve
Rogers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
avengers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
admin</saml2:AttributeValue>
                            </saml2:Attribute>
                        </saml2:AttributeStatement>
                    </saml2:Assertion>
                </RequestedSecurityToken>
                <RequestedAttachedReference>
                    <ns3:SecurityTokenReference xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd" wsse11:TokenType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0">
                        <ns3:KeyIdentifier ValueType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID">
_BDC44EB8593F47D1B213672605113671</ns3:KeyIdentifier>
                    </ns3:SecurityTokenReference>
                </RequestedAttachedReference>
                <RequestedUnattachedReference>
                    <ns3:SecurityTokenReference xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd" wsse11:TokenType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0">
                        <ns3:KeyIdentifier ValueType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID">
_BDC44EB8593F47D1B213672605113671</ns3:KeyIdentifier>
                    </ns3:SecurityTokenReference>
                </RequestedUnattachedReference>
                <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
                    <wsa:EndpointReference xmlns:wsa=
"http://www.w3.org/2005/08/addressing">
                        <wsa:Address>https://server:8993/services/SecurityTokenService</w
sa:Address>
                    </wsa:EndpointReference>
```

```
                </wsp:AppliesTo>
                <Lifetime>
                    <ns2:Created>2013-04-29T18:35:11.444Z</ns2:Created>
                    <ns2:Expires>2013-04-29T19:05:11.444Z</ns2:Expires>
                </Lifetime>
            </RequestSecurityTokenResponse>
        </RequestSecurityTokenResponseCollection>
    </soap:Body>
</soap:Envelope>
```

UsernameToken Bearer SAML Security Token Request/Response

To obtain a SAML assertion to use in secure communication to DDF, a RequestSecurityToken (RST) request has to be made to the STS.

A Bearer SAML assertion is automatically trusted by the endpoint. The client doesn't have to prove it can own that SAML assertion. It is the simplest way to request a SAML assertion, but many endpoints won't accept a KeyType of Bearer.

Request

Explanation

- WS-Addressing header with Action, To, and Message ID
- Valid, non-expired timestamp
- Username Token containing a username and password that the STS will authenticate
- Issued over HTTPS
- KeyType of http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer
- Claims (optional): Some endpoints may require that the SAML assertion include attributes of the user, such as an authenticated user's role, name identifier, email address, etc. If the SAML assertion needs those attributes, the `RequestSecurityToken` must specify which ones to include.

Sample Request

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
        <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd" soap:mustUnderstand="1">
            <wsu:Timestamp wsu:Id="TS-1">
                <wsu:Created>2013-04-29T17:47:37.817Z</wsu:Created>
                <wsu:Expires>2013-04-29T17:57:37.817Z</wsu:Expires>
            </wsu:Timestamp>
            <wsse:UsernameToken wsu:Id="UsernameToken-1">
                <wsse:Username>srogers</wsse:Username>
                <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-username-token-profile-1.0#PasswordText">password1</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
        <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Issue</wsa:Action>
        <wsa:MessageID>uuid:a1bba87b-0f00-46cc-975f-001391658cbe</wsa:MessageID>
        <wsa:To>https://server:8993/services/SecurityTokenService</wsa:To>
    </soap:Header>
    <soap:Body>
        <wst:RequestSecurityToken xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512">
            <wst:SecondaryParameters>
                <t:TokenType xmlns:t="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0</t:TokenType>
                <t:KeyType xmlns:t="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer</t:KeyType>
                <t:Claims xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
xmlns:t="http://docs.oasis-open.org/ws-sx/ws-trust/200512" Dialect=
"http://schemas.xmlsoap.org/ws/2005/05/identity">
                    <!--Add any additional claims you want to grab for the service-->
                    <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/uid"/>
                    <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"/>
                    <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"/>
                    <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"/>
                    <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname"/>
                    <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname"/>
                </t:Claims>
            </wst:SecondaryParameters>
```

```
            <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue</wst:RequestType>
            <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
                <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing">
                    <wsa:Address>https://server:8993/services/QueryService</wsa:Address>
                </wsa:EndpointReference>
            </wsp:AppliesTo>
            <wst:Renewing/>
        </wst:RequestSecurityToken>
    </soap:Body>
</soap:Envelope>
```

Response

This is the response from the STS containing the SAML assertion to be used in subsequent requests to QCRUD endpoints:

The `saml2:Assertion` block contains the entire SAML assertion.

The `Signature` block contains a signature from the STS's private key. The endpoint receiving the SAML assertion will verify that it trusts the signer and ensure that the message wasn't tampered with.

The `AttributeStatement` block contains all the Claims requested.

The `Lifetime` block indicates the valid time interval in which the SAML assertion can be used.

Sample Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <Action xmlns="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/ws-sx/ws-trust/200512/RSTRC/IssueFinal</Action>
        <MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:eee4c6ef-ac10-
4cbc-a53c-13d960e3b6e8</MessageID>
        <To xmlns="http://www.w3.org/2005/08/addressing"
>http://www.w3.org/2005/08/addressing/anonymous</To>
        <RelatesTo xmlns="http://www.w3.org/2005/08/addressing">uuid:a1bba87b-0f00-46cc-
975f-001391658cbe</RelatesTo>
        <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd" soap:mustUnderstand="1">
            <wsu:Timestamp wsu:Id="TS-2">
                <wsu:Created>2013-04-29T17:49:12.624Z</wsu:Created>
                <wsu:Expires>2013-04-29T17:54:12.624Z</wsu:Expires>
            </wsu:Timestamp>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
        <RequestSecurityTokenResponseCollection xmlns="http://docs.oasis-open.org/ws-
sx/ws-trust/200512" xmlns:ns2="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" xmlns:ns3="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd" xmlns:ns4="http://www.w3.org/2005/08/addressing"
xmlns:ns5="http://docs.oasis-open.org/ws-sx/ws-trust/200802">
            <RequestSecurityTokenResponse>
                <TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV2.0</TokenType>
                <RequestedSecurityToken>
                    <saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" ID="_7437C1A55F19AFF22113672577526132" IssueInstant="2013-04-29T17:49:12.613Z"
Version="2.0" xsi:type="saml2:AssertionType">
                        <saml2:Issuer>tokenissuer</saml2:Issuer>
                        <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                            <ds:SignedInfo>
                                <ds:CanonicalizationMethod Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                <ds:SignatureMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                                <ds:Reference URI="#_7437C1A55F19AFF22113672577526132">
                                    <ds:Transforms>
                                        <ds:Transform Algorithm=
"http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                                        <ds:Transform Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#">
```

```
                                        <ec:InclusiveNamespaces xmlns:ec=
"http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="xs"/>
                                    </ds:Transform>
                                </ds:Transforms>
                                <ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
                                <ds:DigestValue>ReOqEbGZlyplW5kqiynXOjPnVEA=</ds:Dige
stValue>
                            </ds:Reference>
                        </ds:SignedInfo>
                        <ds:SignatureValue>X5Kzd54PrKIlGVV2XxzCmWFRzHRoybF7hU6zxbEhSL
MR0AWS9R7Me3epq91XqeOwvIDDbwmE/oJNC7vI0fIw/rqXkx4aZsY5a5nbAs7f+aXF9TGdk82x2eNhNGYpViq0YZJ
fsJ5WSyMtG8w5nRekmDMy9oTLsHG+Y/OhJDEwq58=</ds:SignatureValue>
                        <ds:KeyInfo>
                            <ds:X509Data>
                                <ds:X509Certificate>MIIDmjCCAwOgAwIBAgIBBDANBgkqhkiG9
w0BAQQFADB1MQswCQYDVQQGEwJVUzEQMA4GA1UECBMH
QXJpem9uYTERMA8GA1UEBxMIR29vZHllYXIxEDAOBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0V4
YW1wbGUxEDAOBgNVBAsTB0V4YW1wbGUxCzAJBgNVBAMTAkNBMB4XDTEzMDQwOTE4MzcxMVoXDTIz
MDQwNzE4MzcxMVowgaYxCzAJBgNVBAYTAlVTMRAwDgYDVQQIEwdBcml6b25hMREwDwYDVQQHEwhH
b29keWVhcjEQMA4GA1UEChMHRXhhbXBsZTEQMA4GA1UEChMHRXhhbXBsZTEQMA4GA1UECxMHRXhh
bXBsZTEUMBIGA1UEAxMLdG9rZW5pc3N1ZXIxJjAkBgkqhkiG9w0BCQEWF3Rva2VuaXNzdWVyQGV4
YW1wbGUuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDDfktpA8Lrp9rTfRibKdgtxtN9
uB44diiIqq3JOzDGfDhGLu6mjpuHO1hrKItv42hBOhhmH7lS9ipiaQCIpVfgIG63MB7fa5dBrfGF
G69vFrU1Lfi7IvsVVsNrtAEQljOMmw9sxS3SUsRQX+bD8jq7Uj1hpoF7DdqpV8Kb0COOGwIDAQAB
o4IBBjCCAQIwCQYDVR0TBAIwADAsBglghkgBhvhCAQ0EHxYdT3BlblNTTCBHZW5lcmF0ZWQgQ2Vy
dGlmaWNhdGUwHQYDVR0OBBYEFD1mHviop2Tc4HaNu8yPXR6GqWP1MIGnBgNVHSMEgZ8wgZyAFBcn
en6/j05DzaVwORwrteKc7TZOoXmkdzB1MQswCQYDVQQGEwJVUzEQMA4GA1UECBMHQXJpem9uYTER
MA8GA1UEBxMIR29vZHllYXIxEDAOBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0V4YW1wbGUxEDAO
BgNVBAsTB0V4YW1wbGUxCzAJBgNVBAMTAkNBggkAwXk7OcwO7gwwDQYJKoZIhvcNAQEEBQADgYEA
PiTX5kYXwdhmijutSkrObKpRbQkvkkzcyZlO6VrAxRQ+eFeN6NyuyhgYy5K6l/sIWdaGou5iJOQx
2pQYWx1v8Klyl0W22IfEAXYv/epiO89hpdACryuDJpioXI/X8TAwvRwLKL21Dk3k2b+eyCgA0O++
HM0dPfiQLQ99ElWkv/0=</ds:X509Certificate>
                            </ds:X509Data>
                        </ds:KeyInfo>
                    </ds:Signature>
                    <saml2:Subject>
                        <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified" NameQualifier="http://cxf.apache.org/sts">srogers</saml2:NameID>
                        <saml2:SubjectConfirmation Method=
"urn:oasis:names:tc:SAML:2.0:cm:bearer"/>
                    </saml2:Subject>
                    <saml2:Conditions NotBefore="2013-04-29T17:49:12.614Z"
NotOnOrAfter="2013-04-29T18:19:12.614Z">
                        <saml2:AudienceRestriction>
                            <saml2:Audience>https://server:8993/services/QueryService
</saml2:Audience>
                        </saml2:AudienceRestriction>
```

```
                        </saml2:Conditions>
                        <saml2:AuthnStatement AuthnInstant="2013-04-29T17:49:12.613Z">
                            <saml2:AuthnContext>
                                <saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:a
c:classes:unspecified</saml2:AuthnContextClassRef>
                            </saml2:AuthnContext>
                        </saml2:AuthnStatement>
                        <saml2:AttributeStatement>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
srogers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string"
>srogers@example.com</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
srogers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">Steve
Rogers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
avengers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
admin</saml2:AttributeValue>
                            </saml2:Attribute>
                        </saml2:AttributeStatement>
                    </saml2:Assertion>
                </RequestedSecurityToken>
                <RequestedAttachedReference>
```

```
                    <ns3:SecurityTokenReference xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd" wsse11:TokenType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0">
                        <ns3:KeyIdentifier ValueType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID">
_7437C1A55F19AFF22113672577526132</ns3:KeyIdentifier>
                    </ns3:SecurityTokenReference>
                </RequestedAttachedReference>
                <RequestedUnattachedReference>
                    <ns3:SecurityTokenReference xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd" wsse11:TokenType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0">
                        <ns3:KeyIdentifier ValueType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID">
_7437C1A55F19AFF22113672577526132</ns3:KeyIdentifier>
                    </ns3:SecurityTokenReference>
                </RequestedUnattachedReference>
                <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
                    <wsa:EndpointReference xmlns:wsa=
"http://www.w3.org/2005/08/addressing">
                        <wsa:Address>https://server:8993/services/QueryService</wsa:Addre
ss>
                    </wsa:EndpointReference>
                </wsp:AppliesTo>
                <Lifetime>
                    <ns2:Created>2013-04-29T17:49:12.620Z</ns2:Created>
                    <ns2:Expires>2013-04-29T18:19:12.620Z</ns2:Expires>
                </Lifetime>
            </RequestSecurityTokenResponse>
        </RequestSecurityTokenResponseCollection>
    </soap:Body>
</soap:Envelope>
```

X.509 PublicKey SAML Security Token Request/Response

In order to obtain a SAML assertion to use in secure communication to DDF, a `RequestSecurityToken` (RST) request has to be made to the STS.

An endpoint's policy will specify the type of security token needed. Most of the endpoints that have been used with DDF require a SAML v2.0 assertion with a required KeyType of http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey. This means that the SAML assertion provided by the client to a DDF endpoint must contain a SubjectConfirmation block with a type of "holder-of-key" containing the client's public key. This is used to prove that the client can possess the SAML assertion returned by the STS.

Request

Explanation The STS that comes with DDF requires the following to be in the RequestSecurityToken request in order to issue a valid SAML assertion. See the request block below for an example of how these components should be populated.

- WS-Addressing header containing Action, To, and MessageID blocks
- Valid, non-expired timestamp
- Issued over HTTPS
- TokenType of http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0
- KeyType of http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey
- X509 Certificate as the Proof of Possession or POP. This needs to be the certificate of the client that will be both requesting the SAML assertion and using the SAML assertion to issue a query
- Claims (optional): Some endpoints may require that the SAML assertion include attributes of the user, such as an authenticated user's role, name identifier, email address, etc. If the SAML assertion needs those attributes, the RequestSecurityToken must specify which ones to include.
    - UsernameToken: If Claims are required, the RequestSecurityToken security header must contain a UsernameToken element with a username and password.

Sample Request

```
<soapenv:Envelope xmlns:ns="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
      <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue</wsa:Action>
      <wsa:MessageID>uuid:527243af-94bd-4b5c-a1d8-024fd7e694c5</wsa:MessageID>
      <wsa:To>https://server:8993/services/SecurityTokenService</wsa:To>
      <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu=
"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
         <wsu:Timestamp wsu:Id="TS-17">
            <wsu:Created>2014-02-19T17:30:40.771Z</wsu:Created>
            <wsu:Expires>2014-02-19T19:10:40.771Z</wsu:Expires>
         </wsu:Timestamp>

         <!-- OPTIONAL: Only required if the endpoint that the SAML assertion will be
sent to requires claims. -->
         <wsse:UsernameToken wsu:Id="UsernameToken-16">
            <wsse:Username>pparker</wsse:Username>
            <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordText">password1</wsse:Password>
            <wsse:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-soap-message-security-1.0#Base64Binary">LCTD+5Y7hlWIP6SpsEg9XA==</wsse:Nonce>
            <wsu:Created>2014-02-19T17:30:37.355Z</wsu:Created>
         </wsse:UsernameToken>
      </wsse:Security>
   </soapenv:Header>
   <soapenv:Body>
      <wst:RequestSecurityToken xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512">
         <wst:TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV2.0</wst:TokenType>
         <wst:KeyType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/PublicKey</wst:KeyType>

         <!-- OPTIONAL: Only required if the endpoint that the SAML assertion will be
sent to requires claims. -->
         <wst:Claims Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity" xmlns:ic=
"http://schemas.xmlsoap.org/ws/2005/05/identity">
            <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"/>
            <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"/>
            <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"/>
            <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname"/>
```

```
            <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname"/>
         </wst:Claims>
         <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue</wst:RequestType>
            <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
            <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing">
            <wsa:Address>https://server:8993/services/QueryService</wsa:Address>
            </wsa:EndpointReference>
         </wsp:AppliesTo>
         <wst:UseKey>
            <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
               <ds:X509Data>
                  <ds:X509Certificate>MIIFGDCCBACgAwIBAgICJe0wDQYJKoZIhvcNAQEFBQAwXDELMAk
GA1UEBhMCVVMxGDAWBgNVBAoT
D1UuUy4gR292ZXJubWVudDEMMAoGA1UECxMDRG9EMQwwCgYDVQQLEwNQS0kxFzAVBgNVBAMTDkRP
RCBKSVRDIENBLTI3MB4XDTEzMDUwNzAwMjU0OVoXDTE2MDUwNzAwMjU0OVowaTELMAkGA1UEBhMC
VVMxGDAWBgNVBAoTD1UuUy4gR292ZXJubWVudDEMMAoGA1UECxMDRG9EMQwwCgYDVQQLEwNQS0kx
EzARBgNVBAsTCkNPTlRSQUNUT1IxDzANBgNVBAMTBmNsaWVudDCCASIwDQYJKoZIhvcNAQEBBQAD
ggEPADCCAQoCggEBAOq6L1/jjZ5cyhjhHEbOHr5WQpboKACYbrsn8lg85LGNoAfcwImr9KBmOxGb
ZCxHYIhkW7pJ+kppyH8DbbbDMviIvvdkvrAIU0l8OBRn2wReCBGQ01Imdc3+WzFF2svW75d6wii2ZVd
eMvUO15p/pAD/sdIfXmAfyu8+tqtiO8KVZGkTnlg3AMzfeSrkci5UHMVWj0qUSuzLk9SAg/9STgb
Kf2xBpHUYecWFSB+dTpdZN2pC85tj9xIoWGh5dFWG1fPcYRgzGPxsybiGOylbJ7rHDJuL7IIIyx5
EnkCuxmQwoQ6XQAhiWRGyPlY08w1LZixI2v+Cv/ZjUfIHv49I9P4Mt8CAwEAAaOCAdUwggHRMB8G
A1UdIwQYMBaAFCMUNCBNXy43NZLBBlnDjDplNZJoMB0GA1UdDgQWBBRPGiX6zZzKTqQSx/tjg6hx
9opDoTAOBgNVHQ8BAf8EBAMCBaAwgdoGA1UdHwSB0jCBzzA2oDSgMoYwaHR0cDovL2NybC5nZHMu
bml0LmRpc2EubWlsL2NybC9ET0RKSVRDQ0FfMjcuY3JsMIGUoIGRoIGOhoGLbGRhcDovL2NybC5n
ZHMubml0LmRpc2EubWlsL2NuJTNkRE9EJTIwSklUQyUyMENBLTI3JTJjb3UlM2RQS0klMmNvdSUz
ZERvRCUyY28lM2RVLlMuJTIwR292ZXJubWVudCUyY2MlM2RVUz9jZXJ0aWZpY2F0ZXJldm9jYXRp
b25saXN0O2JpbmFyeTAjBgNVHSAEHDAaMAsGCWCGSAFlAgELBTALBglghkgBZQIBCxIwfQYIKwYB
BQUHAQEEcTBvMD0GCCsGAQUFBzAChjFodHRwOi8vY3JsLmdkcy5uaXQuZGlzYS5taWwvc2lnbi9E
T0RKSVRDQ0FfMjcuY2VyMC4GCCsGAQUFBzABhiJodHRwOi8vb2NzcC5uc24wLnJjdnMubml0LmRp
c2EubWlsMA0GCSqGSIb3DQEBBQUAA4IBAQCGUJPGh4iGCbr2xCMqCq04SFQ+iaLmTIFAxZPFvup1
4E9Ir6CSDalpF9eBx9fS+Z2xuesKyM/g3YqWU1LtfWGRRIxzEujaC4YpwHuffkx9QqkwSkXXIsim
EhmzSgzxnT4Q9X8WwalqVYOfNZ6sSLZ8qPPFrLHkkw/zIFRzo62wXLu0tfcpOr+iaJBhyDRinIHr
hwtE3xo6qQRRWlO3/clC4RnTev1crFVJQVBF3yfpRu8udJ2SOGdqU0vjUSu1h7aMkYJMHIu08Whj
8KASjJBFeHPirMV1oddJ5ydZCQ+Jmnpbwq+XsCxg1LjC4dmbjKVr9s4QK+/JLNjxD8IkJiZE</ds:X509Certific
ate>
               </ds:X509Data>
            </ds:KeyInfo>
         </wst:UseKey>
      </wst:RequestSecurityToken>
   </soapenv:Body>
</soapenv:Envelope>
```

Response

Explanation This is the response from the STS containing the SAML assertion to be used in subsequent requests to QCRUD endpoints.

The `saml2:Assertion` block contains the entire SAML assertion.

The `Signature` block contains a signature from the STS's private key. The endpoint receiving the SAML assertion will verify that it trusts the signer and ensure that the message wasn't tampered with.

The `SubjectConfirmation` block contains the client's public key, so the server can verify that the client has permission to hold this SAML assertion. The `AttributeStatement` block contains all of the claims requested.

Sample Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header>
      <Action xmlns="http://www.w3.org/2005/08/addressing">http://docs.oasis-open.org/ws-
sx/ws-trust/200512/RSTRC/IssueFinal</Action>
      <MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:b46c35ad-3120-
4233-ae07-b9e10c7911f3</MessageID>
      <To xmlns="http://www.w3.org/2005/08/addressing"
>http://www.w3.org/2005/08/addressing/anonymous</To>
      <RelatesTo xmlns="http://www.w3.org/2005/08/addressing">uuid:527243af-94bd-4b5c-
a1d8-024fd7e694c5</RelatesTo>
      <wsse:Security soap:mustUnderstand="1" xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu=
"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
         <wsu:Timestamp wsu:Id="TS-90DBA0754E55B4FE7013928310431357">
            <wsu:Created>2014-02-19T17:30:43.135Z</wsu:Created>
            <wsu:Expires>2014-02-19T17:35:43.135Z</wsu:Expires>
         </wsu:Timestamp>
      </wsse:Security>
   </soap:Header>
   <soap:Body>
      <ns2:RequestSecurityTokenResponseCollection xmlns="http://docs.oasis-open.org/ws-
sx/ws-trust/200802" xmlns:ns2="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
xmlns:ns3="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd" xmlns:ns4="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" xmlns:ns5="http://www.w3.org/2005/08/addressing">
         <ns2:RequestSecurityTokenResponse>
            <ns2:TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV2.0</ns2:TokenType>
            <ns2:RequestedSecurityToken>
               <saml2:Assertion ID="_90DBA0754E55B4FE7013928310431176" IssueInstant=
"2014-02-19T17:30:43.117Z" Version="2.0" xsi:type="saml2:AssertionType" xmlns:saml2=
"urn:oasis:names:tc:SAML:2.0:assertion" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                  <saml2:Issuer>tokenissuer</saml2:Issuer>
                  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                     <ds:SignedInfo>
                        <ds:CanonicalizationMethod Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#"/>
                        <ds:SignatureMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                        <ds:Reference URI="#_90DBA0754E55B4FE7013928310431176">
                           <ds:Transforms>
                              <ds:Transform Algorithm=
"http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                              <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#">
```

```
                                 <ec:InclusiveNamespaces PrefixList="xs" xmlns:ec=
"http://www.w3.org/2001/10/xml-exc-c14n#"/>
                              </ds:Transform>
                           </ds:Transforms>
                           <ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
                           <ds:DigestValue>/bEGqsRGHVJbx298WPmGd8I53zs=</ds:DigestValue>
                        </ds:Reference>
                     </ds:SignedInfo>
                     <ds:SignatureValue>
mYR7w1/dnuh8Z7t9xjCb4XkYQLshj+UuYlGOuTwDYsUPcS2qI0nAgMD1VsDP7y1fDJxeqsq7HYhFKsnqRfebMM4WL
H1D/lJ4rD4UO+i9l3tuiHml7SN24WM1/bOqfDUCoDqmwG8afUJ3r4vmTNPxfwfOss8BZ/8ODgZzm08ndlkxDfvcN7
OrExbV/3/45JwF/MMPZoqvi2MJGfX56E9fErJNuzezpWnRqPOlWPxyffKMAlVaB9zF6gvVnUqcW2k/Z8X9lN7O5jo
uBI281ZnIfsIPuBJERFtYNVDHsIXM1pJnrY6FlKIaOsi55LQu3Ruir/n82pU7BT5aWtxwrn7akBg==
</ds:SignatureValue>
                     <ds:KeyInfo>
                        <ds:X509Data>
                           <ds:X509Certificate>MIIFHTCCBAWgAwIBAgICJe8wDQYJKoZIhvcNAQEFBQ
AwXDELMAkGA1UEBhMCVVMxGDAWBgNVBAoT
D1UuUy4gR292ZXJubWVudDEMMAoGA1UECxMDRG9EMQwwCgYDVQQLEwNQS0kxFzAVBgNVBAMTDkRP
RCBKSVRDIENBLTI3MB4XDTEzMDUwNzAwMjYzN1oXDTE2MDUwNzAwMjYzN1owbjELMAkGA1UEBhMC
VVMxGDAWBgNVBAoTD1UuUy4gR292ZXJubWVudDEMMAoGA1UECxMDRG9EMQwwCgYDVQQLEwNQS0kx
EzARBgNVBAsTCkNPTlRSQUNUT1IxFDASBgNVBAMTC3Rva2VuaXNzdWVyMIIBIjANBgkqhkiG9w0B
AQEFAAOCAQ8AMIIBCgKCAQEAx01/U4M1wG+wL1JxX2RL1glj101FkJXMk3KFt3zD//N8x/Dcwwvs
ngCQjXrV6YhbB2V7scHwnThPv3RSwYYiO62z+g6ptfBbKGGBLSZOzLe3fyJR4RxblFKsELFgPHfX
vgUHS/keG5uSRk9S/Okqps/yxKB7+ZlxeFxsIz5QywXvBpMiXtc2zF+M7BsbSIdSx5LcPcDFBwjF
c66rE3/y/25VMht9EZX1QoKr7f8rWD4xgd5J6DYMFWEcmiCz4BDJH9sfTw+n1P+CYgrhwslWGqxt
cDME9t6SWR3GLT4Sdtr8ziIM5uUteEhPIV3rVC3/u23JbYEeS8mpnp0bxt5eHQIDAQABo4IB1TCC
AdEwHwYDVR0jBBgwFoAUIxQ0IE1fLjc1ksEGWcOMOmU1kmgwHQYDVR0OBBYEFGBjdkdey+bMHMhC
Z7gwiQ/mJf5VMA4GA1UdDwEB/wQEAwIFoDCB2gYDVR0fBIHSMIHPMDagNKAyhjBodHRwOi8vY3Js
Lmdkcy5uaXQuZGlzYS5taWwvY3JsL0RPREpJVENDQV8yNy5jcmwwgZSggZGggY6GgYtsZGFwOi8v
Y3JsLmdkcy5uaXQuZGlzYS5taWwvY24lM2RET0QlMjBKSVRDJTIwQ0EtMjclMmNvdSUzZFBLSSUy
Y291JTNkRG9EJTJjbyUzZFUuUy4lMjBHb3Zlcm5tZW50JTJjYyUzZFVTP2NlcnRpZmljYXRlcmV2
b2NhdGlvbmxpc3Q7YmluYXJ5MCMGA1UdIAQcMBowCwYJYIZIAWUCAQsFMAsGCWCGSAFlAgELEjB9
BggrBgEFBQcBAQRxMG8wPQYIKwYBBQUHMAKGMWh0dHA6Ly9jcmwuZ2RzLm5pdC5kaXNhLm1pbC9z
aWduL0RPREpJVENDQV8yNy5jZXIwLgYIKwYBBQUHMAGGImh0dHA6Ly9vY3NwLm5zbjAucmN2cy5u
aXQuZGlzYS5taWwwDQYJKoZIhvcNAQEFBQADggEBAIHZQTINU3bMpJ/PkwTYLWPmwCqAYgEUzSYx
bNcVY5MWD8b4XCdw5nM3GnFlOqr4IrHeyyOzsEbIebTe3bv0l1pHx0Uyj059nAhx/AP8DjVtuRU1
/Mp4b6uJ/4yaoMjIGceqBzHqhHIJinG0Y2azua7eM9hVbWZsa912ihbiupCq22mYuHFP7NUNzBvV
j03YUcsy/sES5sRx9Rops/CBN+LUUYOdJOxYWxo8oAbtF8ABE5ATLAwqz4ttsToKPUYh1sxdx5Ef
APeZ+wYDmMu4OfLckwnCKZgkEtJOxXpdIJHY+VmyZtQSB0LkR5toeH/ANV4259Ia5ZT8h2/vIJBg
6B4=</ds:X509Certificate>
                        </ds:X509Data>
                     </ds:KeyInfo>
                  </ds:Signature>
                  <saml2:Subject>
                     <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified" NameQualifier="http://cxf.apache.org/sts">pparker</saml2:NameID>
```

```
                    <saml2:SubjectConfirmation Method=
"urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
                        <saml2:SubjectConfirmationData xsi:type=
"saml2:KeyInfoConfirmationDataType">
                            <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                               <ds:X509Data>
                                  <ds:X509Certificate>MIIFGDCCBACgAwIBAgICJe0wDQYJKoZIhvcN
AQEFBQAwXDELMAkGA1UEBhMCVVMxGDAWBgNVBAoT
D1UuUy4gR292ZXJubWVudDEMMAoGA1UECxMDRG9EMQwwCgYDVQQLEwNQS0kxFzAVBgNVBAMTDkRP
RCBKSVRDIENBLTI3MB4XDTEzMDUwNzAwMjU0OVoXDTE2MDUwNzAwMjU0OVowaTELMAkGA1UEBhMC
VVMxGDAWBgNVBAoTD1UuUy4gR292ZXJubWVudDEMMAoGA1UECxMDRG9EMQwwCgYDVQQLEwNQS0kx
EzARBgNVBAsTCkNPTlRSQUNUT1IxDzANBgNVBAMTBmNsaWVudDCCASIwDQYJKoZIhvcNAQEBBQAD
ggEPADCCAQoCggEBAOq6L1/jjZ5cyhjhHEbOHr5WQpboKACYbrsn8lg85LGNoAfcwImr9KBmOxGb
ZCxHYIhkW7pJ+kppyH8bbbviIvvdkvrAIU0l8OBRn2wReCBGQ01Imdc3+WzFF2svW75d6wii2ZVd
eMvUO15p/pAD/sdIfXmAfyu8+tqtiO8KVZGkTnlg3AMzfeSrkci5UHMVWj0qUSuzLk9SAg/9STgb
Kf2xBpHUYecWFSB+dTpdZN2pC85tj9xIoWGh5dFWG1fPcYRgzGPxsybiGOylbJ7rHDJuL7IIIyx5
EnkCuxmQwoQ6XQAhiWRGyPlY08w1LZixI2v+Cv/ZjUfIHv49I9P4Mt8CAwEAAaOCAdUwggHRMB8G
A1UdIwQYMBaAFCMUNCBNXy43NZLBBlnDjDplNZJoMB0GA1UdDgQWBBRPGiX6zZzKTqQSx/tjg6hx
9opDoTAOBgNVHQ8BAf8EBAMCBaAwgdoGA1UdHwSB0jCBzzA2oDSgMoYwaHR0cDovL2NybC5nZHMu
bml0LmRpc2EubWlsL2NybC9ET0RKSVRDQ0FfMjcuY3JsMIGUoIGRoIGOhoGLbGRhcDovL2NybC5n
ZHMubml0LmRpc2EubWlsL2NuJTNkRE9EJTIwSklUQyUyMENBLTI3JTJjb3UlM2RQS0klMmNvdSUz
ZERvRCUyY28lM2RVLlMuJTIwR292ZXJubWVudCUyY2MlM2RVUz9jZXJ0aWZpY2F0ZXJldm9jYXRp
b25saXN0O2JpbmFyeTAjBgNVHSAEHDAaMAsGCWCGSAFlAgELBTALBglghkgBZQIBCxIwfQYIKwYB
BQUHAQEEcTBvMD0GCCsGAQUFBzAChjFodHRwOi8vY3JsLmdkcy5uaXQuZGlzYS5taWwvc2lnbi9E
T0RKSVRDQ0FfMjcuY2VyMC4GCCsGAQUFBzABhiJodHRwOi8vb2NzcC5uc24wLnJjdnMubml0LmRp
c2EubWlsMA0GCSqGSIb3DQEBBQUAA4IBAQCGUJPGh4iGCbr2xCMqCq04SFQ+iaLmTIFAxZPFvup1
4E9Ir6CSDalpF9eBx9fS+Z2xuesKyM/g3YqWU1LtfWGRRIxzEujaC4YpwHuffkx9QqkwSkXXIsim
EhmzSgzxnT4Q9X8WwalqVYOfNZ6sSLZ8qPPFrLHkkw/zIFRzo62wXLu0tfcpOr+iaJBhyDRinIHr
hwtE3xo6qQRRWlO3/clC4RnTev1crFVJQVBF3yfpRu8udJ2SOGdqU0vjUSu1h7aMkYJMHIu08Whj
8KASjJBFeHPirMV1oddJ5ydZCQ+Jmnpbwq+XsCxg1LjC4dmbjKVr9s4QK+/JLNjxD8IkJiZE</ds:X509Certific
ate>
                               </ds:X509Data>
                            </ds:KeyInfo>
                        </saml2:SubjectConfirmationData>
                    </saml2:SubjectConfirmation>
                </saml2:Subject>
                <saml2:Conditions NotBefore="2014-02-19T17:30:43.119Z" NotOnOrAfter=
"2014-02-19T18:00:43.119Z"/>
                <saml2:AuthnStatement AuthnInstant="2014-02-19T17:30:43.117Z">
                    <saml2:AuthnContext>
                        <saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classe
s:unspecified</saml2:AuthnContextClassRef>
                    </saml2:AuthnContext>
                </saml2:AuthnStatement>

                <!-- This block will only be included if Claims were requested in the
RST. -->
                <saml2:AttributeStatement>
```

```
                    <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">
pparker</saml2:AttributeValue>
                    </saml2:Attribute>
                    <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">
pparker@example.com</saml2:AttributeValue>
                    </saml2:Attribute>
                    <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">
pparker</saml2:AttributeValue>
                    </saml2:Attribute>
                    <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">Peter
Parker</saml2:AttributeValue>
                    </saml2:Attribute>
                    <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">
users</saml2:AttributeValue>
                    </saml2:Attribute>
                    <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">
users</saml2:AttributeValue>
                    </saml2:Attribute>
                    <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">
avengers</saml2:AttributeValue>
                    </saml2:Attribute>
                    <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">
admin</saml2:AttributeValue>
                    </saml2:Attribute>
```

```
                  </saml2:AttributeStatement>
               </saml2:Assertion>
            </ns2:RequestedSecurityToken>
            <ns2:RequestedAttachedReference>
               <ns4:SecurityTokenReference wsse11:TokenType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0" xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd">
                  <ns4:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-wss-
saml-token-profile-1.1#SAMLID">_90DBA0754E55B4FE7013928310431176</ns4:KeyIdentifier>
               </ns4:SecurityTokenReference>
            </ns2:RequestedAttachedReference>
            <ns2:RequestedUnattachedReference>
               <ns4:SecurityTokenReference wsse11:TokenType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0" xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd">
                  <ns4:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-wss-
saml-token-profile-1.1#SAMLID">_90DBA0754E55B4FE7013928310431176</ns4:KeyIdentifier>
               </ns4:SecurityTokenReference>
            </ns2:RequestedUnattachedReference>
            <ns2:Lifetime>
               <ns3:Created>2014-02-19T17:30:43.119Z</ns3:Created>
               <ns3:Expires>2014-02-19T18:00:43.119Z</ns3:Expires>
            </ns2:Lifetime>
         </ns2:RequestSecurityTokenResponse>
      </ns2:RequestSecurityTokenResponseCollection>
   </soap:Body>
</soap:Envelope>
```

2.11. Expansion Service

The Expansion Service and its corresponding expansion-related commands provide an easy way for developers to add expansion capabilities to DDF during user attribute and metadata card processing. In addition to these two defined uses of the expansion service, developers are free to utilize the service in their own implementations.

Each instance of the expansion service consists of a collection of rule sets. Each rule set consists of a key value and its associated set of rules. Callers of the expansion service provide a key and an original value to be expanded. The expansion service then looks up the set of rules for the specified key. The expansion service then cumulatively applies each of the rules in the set starting with the original value, with the resulting set of values being returned to the caller.

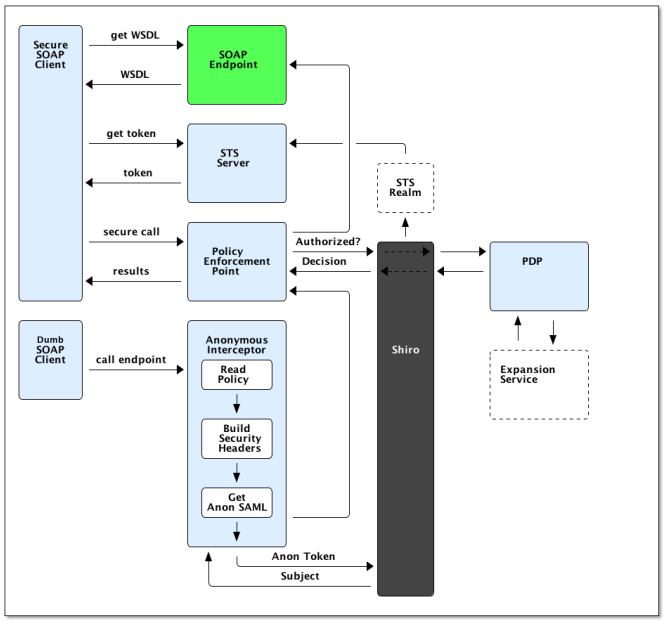
Key (Attribute)

The examples below use the following collection of rule sets:

Key (Attribute)

Note that the rules listed for each key are processed in order, so they may build upon each other, i.e., a new value from the new replacement string may be expanded by a subsequent rule.

2.12. Securing SOAP



2.12.1. SOAP Secure Client

When calling to an endpoint from a SOAP secure client, it first requests the WSDL from the endpoint and the SOAP endpoint returns the WSDL. The client then calls to STS for authentication token to proceed. If the client receives the token, it makes a secure call to the endpoint and receives results.

2.12.2. Dumb SOAP Client

If calling endpoint from a non-secure client, at the point the of the initial call, the Guest Interceptor catches the request and prepares it to be accepted by the endpoint.

First, the interceptor reads the configured policy, builds a security header, and gets an anonymous SAML assertion. Using this, it makes a `getSubject` call which is sent through Shiro to the STS realm. Upon success, the STS realm returns the subject and the call is made to the endpoint.

3. Extending DDF Admin

Version: 2.9.1

This section supports developers creating extensions of the existing DDF Admin application.

3.1. DDF Admin Whitelist

The following packages have been exported by the DDF Admin application and are approved for use by third parties:

- `org.codice.ddf.ui.admin.api`
- `org.codice.ddf.ui.admin.api.module`
- `org.codice.ddf.ui.admin.api.plugin`
- `org.codice.ddf.admin.configuration.plugin`
- `org.codice.ddf.admin.application.service`
- `org.codice.ddf.admin.application.plugin`

4. Extending DDF Catalog

Version: 2.9.1

The DDF Catalog provides a framework for storing, searching, processing, and transforming information. Clients typically perform query, create, read, update, and delete (QCRUD) operations against the Catalog. At the core of the Catalog functionality is the Catalog Framework, which routes all requests and responses through the system, invoking additional processing per the system configuration.

This guide supports developers creating extensions of the existing framework.

4.1. Whitelist

The following packages have been exported by the DDF Catalog application and are approved for use by third parties:

- `ddf.camel.component.catalog`
- `ddf.catalog`
- `ddf.catalog.cache`
- `ddf.catalog.data`
- `ddf.catalog.data.metacardtype`
- `ddf.catalog.event`
- `ddf.catalog.federation`
- `ddf.catalog.federation.impl`
- `ddf.catalog.filter`
- `ddf.catalog.filter.delegate`
- `ddf.catalog.impl.filter`
- `ddf.catalog.operation`
- `ddf.catalog.plugin`
- `ddf.catalog.plugin.groomer`
- `ddf.catalog.pubsub`
- `ddf.catalog.pubsub.tracker`
- `ddf.catalog.resource`
- `ddf.catalog.resource.data`

- ddf.catalog.resource.impl
- ddf.catalog.resourceretriever
- ddf.catalog.service
- ddf.catalog.source
- ddf.catalog.transform
- ddf.catalog.transformer.api
- ddf.catalog.transformer.metacard.geojson
- ddf.catalog.util
- ddf.catalog.validation
- ddf.common
- ddf.geo.formatter
- ddf.util
- org.codice.ddf.endpoints
- org.codice.ddf.endpoints.rest
- org.codice.ddf.endpoints.rest.action
- org.codice.ddf.opensearch.query
- org.codice.ddf.opensearch.query.filter

4.2. Catalog Architecture

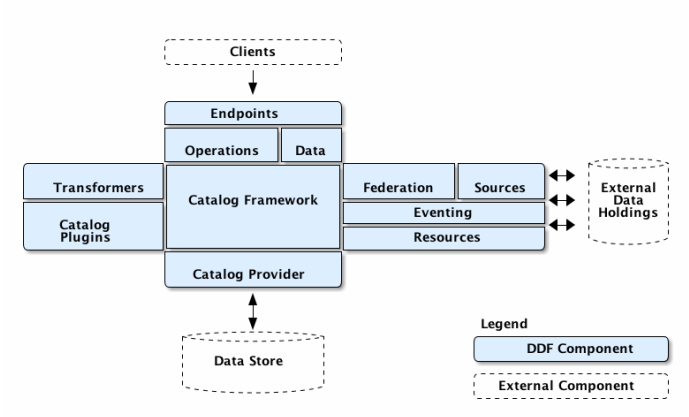


Figure 2. Catalog Architecture

4.3. Catalog Application Services

As an OSGi system, DDF does Intra-module conversations via services. The following summarizes DDF

internal services within the Catalog application.

4.3.1. Catalog Framework

The `CatalogFramework` is the routing mechanism between catalog components that provides integration points for the Catalog Plugins. An endpoint invokes the active Catalog Framework, which calls any configured Pre-query or Pre-ingest plug-ins. The selected federation strategy calls the active Catalog Provider and any connected or federated sources. Then, any Post-query or Post-ingest plug-ins are invoked. Finally, the appropriate response is returned to the calling endpoint.

4.3.2. Sources

A source is a system consisting of a catalog containing Metacards.

4.3.3. CatalogProvider

The Catalog Provider is an API is used to interact with data providers, such as files systems or databases, to query, create, update, or delete data. The provider also translates between DDF objects and native data formats.

4.3.4. `ConnectedSource`

A Connected Source is a local or remote source that is always included in every local and enterprise query, but is hidden from being queried individually.

4.3.5. `FederatedSource`

A Federated Source is a remote source that can be optionally included or excluded from queries.

4.3.6. Plugins

Plugins are additional tools to use to add additional business logic at certain points, depending on the type of plugin. Plugins can be designed to run before or after certain processes. They are often used for validation, optimization, or logging.

"Pre-" Plugins

These plugins are executed before an action is taken.

Plugin	Description
Pre-IngestPlugin	Performs any changes to a resource prior to ingesting it.
Pre-Query Plugin	Performs any changes to query before executing.
Pre-Resource Plugin	Performs any changes to a resource associated with a metacard prior to download.

Plugin	Description
Pre-Subscription Plugin	Performs any changes before creating a subscription.
Pre-Delivery Plugin	Performs any changes before delivered a subscribed event.

“Post-“ Plugins

Plugin	Description
Post-Ingest Plugin	Performs actions after ingest is completed.
Post-Query Plugin	Performs any changes to response after query completes.
Post-Get Resource Plugin	performs any changes to a resource after download

4.3.7. Transformers

Transformers are used to alter the format of a resource or its metadata to or from the catalog’s metacard format

Transformer	Description
Input Transformers	create metacards from input.
Metacard Transformers	translates a metacard from catalog metadata to a specific data format.
Query Response Transformers	translates a list of Result objects to a desired format.

4.4. Catalog Development Fundamentals

This section introduces the fundamentals of working with the Catalog API the OGC Filter for Queries.

4.4.1. Simple Catalog API Implementations

The Catalog API implementations, which are denoted with the suffix of `Impl` on the Java file names, have multiple purposes and uses.

- First, they provide a good starting point for other developers to extend functionality in the framework. For instance, extending the `MetacardImpl` allows developers to focus less on the inner workings of DDF and more on the developer’s intended purposes and objectives.
- Second, the Catalog API Implementations display the proper usage of an interface and an interface’s

intentions. Also, they are good code examples for future implementations. If a developer does not want to extend the simple implementations, the developer can at least have a working code reference to base future development.

4.4.2. Use of the Whiteboard Design Pattern

The DDF Catalog makes extensive use of the Whiteboard Design Pattern. Catalog Components are registered as services in the OSGi Service Registry, and the Catalog Framework or any other clients tracking the OSGi Service Registry are automatically notified by the OSGi Framework of additions and removals of relevant services.

The Whiteboard Design Pattern is a common OSGi technique that is derived from a technical whitepaper provided by the OSGi Alliance in 2004. It is recommended to use the Whiteboard pattern over the Listener pattern in OSGi because it provides less complexity in code (both on the client and server sides), fewer deadlock possibilities than the Listener pattern, and closely models the intended usage of the OSGi framework.

4.4.3. Working with Queries

Clients use `DDF.catalog.operation.Query` objects to describe which metacards are needed from Sources. Query objects have two major components:

- Filter
- Query Options

A Source uses the Filter criteria constraints to find the requested set of metacards within its domain of metacards. The Query Options are used to further restrict the Filter's set of requested metacards.

Query Options

Option	Description
`StartIndex`	1-based index that states which metacard the Source should return first out of the requested metacards.
`PageSize`	Represents the maximum amount of metacards the Source should return.
`SortBy`	Determines how the results are sorted and on which property.
`RequestsTotalResultsCount`	Determines whether the total number of results should be returned.
`TimeoutMillis`	The amount of time in milliseconds before the query is to be abandoned.

Creating a query

The easiest way to create a Query is to use `DDF.catalog.operation.QueryImpl` object. It is first necessary to create an OGC Filter object then set the Query Options after `QueryImpl` has been constructed.

QueryImpl Example 1

```
/*
  Builds a query that requests a total results count and
  that the first record to be returned is the second record found from
  the requested set of metacards.
 */

 String property = ...;

 String value = ...;

 org.geotools.filter.FilterFactoryImpl filterFactory = new FilterFactoryImpl() ;

 QueryImpl query = new QueryImpl( filterFactory.equals(filterFactory.property(property),

filterFactory.literal(value))) ;

 query.setStartIndex(2) ;

 query.setRequestsTotalResultsCount(true);
```

Evaluating a query

Every Source must be able to evaluate a Query object. Nevertheless, each Source could evaluate the Query differently depending on what that Source supports as to properties and query capabilities. For instance, a common property all Sources understand is id, but a Source could possibly store frequency values under the property name "frequency." Some Sources may not support frequency property inquiries and will throw an error stating it cannot interpret the property. In addition, some Sources might be able to handle spatial operations, while others might not. A developer should consult a Source's documentation for the limitations, capabilities, and properties that a Source can support.

4.5. Working with Filters

An OGC Filter is a Open Geospatial Consortium (OGC) standard (http://www.opengeospatial.org/standards/filter) that describes a query expression in terms of Extensible Markup Language (XML) and key-value pairs (KVP). The DDF Catalog Framework does not use the XML representation of the OGC Filter standard. DDF instead utilizes the Java implementation provided by Geotools (http://geotools.org/). Geotools provides Java equivalent classes for OGC Filter XML elements. Geotools originally provided the standard Java classes for the OGC Filter Encoding 1.0

under the package name `org.opengis.filter`. The same package name is used today and is currently used by DDF. Java developers do not parse or view the XML representation of a Filter in DDF. Instead, developers use only the Java objects to complete query tasks.

Note that the DDF.catalog.operation.Query interface extends the org.opengis.filter.Filter interface, which means that a Query object is an OGC Java Filter with Query Options.

A Query is an OGC Filter

```
public interface Query extends Filter
```

4.5.1. Using Filters

4.5.2. FilterBuilder API

To abstract developers from the complexities of working with the Filter interface directly and implementing the DDF Profile of the Filter specification, the DDF Catalog includes an API, primarily in `DDF.filter`, to build Filters using a fluent API.

To use the FilterBuilder API, an instance of `DDF.filter.FilterBuilder` should be used via the OSGi registry. Typically, this will be injected via a dependency injection framework. Once an instance of `FilterBuilder` is available, methods can be called to create and combine Filters.

TIP	The fluent API is best accessed using an IDE that supports code-completion. For additional details, refer to the Catalog API Javadoc.

4.5.3. Boolean Operators

`FilterBuilder.allOf(Filter )`

creates a new Filter that requires all provided Filters are satisfied (Boolean AND), either from a List or Array of Filter instances.

`FilterBuilder.anyOf(Filter )`

creates a new Filter that requires all provided Filters are satisfied (Boolean OR), either from a List or Array of Filter instances.

`FilterBuilder.not(Filter filter)`

creates a new Filter that requires the provided Filter must not be match (Boolean NOT).

Attribute

`FilterBuilder.attribute(String attributeName)`

begins a fluent API for creating an Attribute-based Filter, i.e., a Filter that matches on Metacards with Attributes of a particular value.

XPath

`FilterBuilder.xpath(String xpath)`

begins a fluent API for creating an XPath-based Filter, i.e., a Filter that matches on Metacards with Attributes of type XML that match when evaluating a provided XPath selector.

Contextual Operators

```
FilterBuilder.attribute(attributeName).is().like().text(String contextualSearchPhrase);
FilterBuilder.attribute(attributeName).is().like().caseSensitiveText(StringcaseSensitiveC
ontextualSearchPhrase);
FilterBuilder.attribute(attributeName).is().like().fuzzyText(String fuzzySearchPhrase);
```

Directly Implementing the Filter (Advanced)

WARNING	Implementing the Filter interface directly is only for extremely advanced use cases and is highly discouraged. Instead, use of the DDF-specific `FilterBuilder` API is recommended.

Developers create a `Filter` object in order to filter or constrain the amount of records returned from a `Source`. The OGC Filter Specification has several types of filters that can be combined in a tree-like structure to describe the set of metacards that should be returned.

Categories of Filters

- Comparison Operators
- Logical Operators
- Expressions
- Literals
- Functions
- Spatial Operators
- Temporal Operators

Units of Measure

According to the OGC Filter Specifications: 09-026r1 and OGC Filter Specifications: 04-095, units of measure can be expressed as a URI. To fulfill that requirement, DDF utilizes the Geotools class `org.geotools.styling.UomOgcMapping` for spatial filters requiring a standard for units of measure for scalar distances. Essentially, the `UomOgcMapping` maps the OGC Symbology Encoding standard URIs to Java Units. This class provides three options for units of measure:

- FOOT
- METRE
- PIXEL

DDF only supports FOOT and METRE since they are the most applicable to scalar distances.

Creating Filters

The common way to create a `Filter` is to use the Geotools `FilterFactoryImpl` object, which provides Java implementations for the various types of filters in the Filter Specification. Examples are the easiest way to understand how to properly create a `Filter` and a `Query`.

NOTE | Refer to the Geotools javadoc for more information on `FilterFactoryImpl`.

The example below illustrates creating a query, and thus an OGC Filter, that does a case-insensitive search for the phrase "mission" in the entire metacard's text. Note that the OGC `PropertyIsLike` Filter is used for this simple contextual query.

Example Creating-Filters-1

Simple Contextual Search

```
org.opengis.filter.FilterFactory filterFactory = new FilterFactoryImpl() ;
boolean isCaseSensitive = false ;

String wildcardChar = "*" ; // used to match zero or more characters
String singleChar = "?" ; // used to match exactly one character
String escapeChar = "\\" ; // used to escape the meaning of the wildCard, singleChar,
and the escapeChar itself

String searchPhrase = "mission" ;
org.opengis.filter.Filter propertyIsLikeFilter =
    filterFactory.like(filterFactory.property(Metacard.ANY_TEXT), searchPhrase,
wildcardChar, singleChar, escapeChar, isCaseSensitive);
DDF.catalog.operation.QueryImpl query = new QueryImpl( propertyIsLikeFilter );
```

The example below illustrates creating an absolute temporal query, meaning the query is searching for Metacards whose modified timestamp occurred during a specific time range. Note that this query uses the `During` OGC Filter for an absolute temporal query.

Example Creating-Filters-2

Absolute Temporal Search

```
org.opengis.filter.FilterFactory filterFactory = new FilterFactoryImpl() ;
org.opengis.temporal.Instant startInstant = new org.geotools.temporal.object
.DefaultInstant(new DefaultPosition(start));

org.opengis.temporal.Instant endInstant = new org.geotools.temporal.object.
DefaultInstant(new DefaultPosition(end));

org.opengis.temporal.Period period =  new org.geotools.temporal.object.DefaultPeriod
(startInstant, endInstant);

String property = Metacard.MODIFIED ; // modified date of a metacard

org.opengis.filter.Filter filter = filterFactory.during( filterFactory.property(property
), filterFactory.literal(period)  );

DDF.catalog.operation.QueryImpl query = new QueryImpl(filter) ;
```

Contextual Searches

Most contextual searches can be expressed using the `PropertyIsLike` filter. The special haracters that have meaning in a `PropertyIsLike` filter are the wildcard, single wildcard, and escape characters (see Example Creating-Filters-1).

`PropertyIsLike` **Special Characters**

Character	Description
Wildcard	Matches zero or more characters.
Single Wildcard	Matches exactly one character.
Escape	Escapes the meaning of the Wildcard, Single Wildcard, and the Escape character itself

Characters and words, such as `AND`, `&`, `and`, `OR`, `|`, `or`, `NOT`, `~`, `not`, `{`, and `}`, are treated as literals in a `PropertyIsLike` filter. In order to create equivalent logical queries, a developer must instead use the Logical Operator filters {`AND`, `OR`, `NOT`}. The Logical Operator filters can be combined together with `PropertyIsLike` filters to create a tree that represents the search phrase expression.

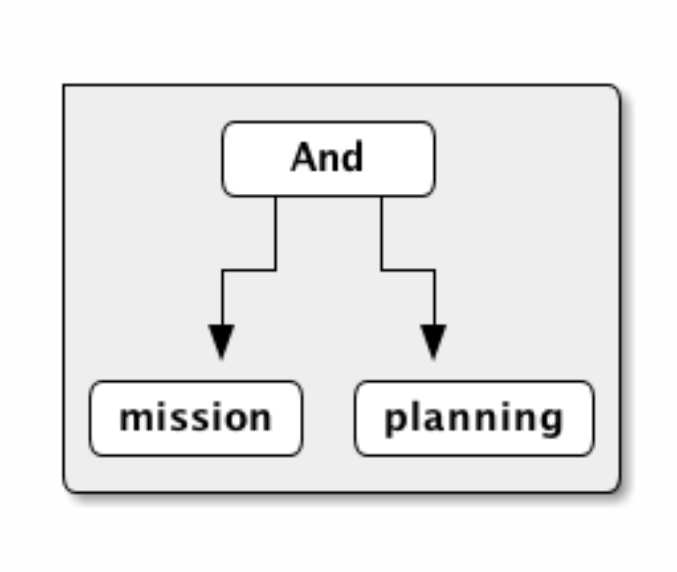
Example Creating-Filters-3

Creating the search phrase "mission and planning"

```
org.opengis.filter.FilterFactory filterFactory = new FilterFactoryImpl() ;

boolean isCaseSensitive = false ;

String wildcardChar = "*" ; // used to match zero or more characters
String singleChar = "?" ; // used to match exactly one character
String escapeChar = "\\" ; // used to escape the meaning of the wildCard, singleChar, and
the escapeChar itself

Filter filter =
    filterFactory.and(
       filterFactory.like(filterFactory.property(Metacard.METADATA), "mission" ,
wildcardChar, singleChar, escapeChar, isCaseSensitive),
       filterFactory.like(filterFactory.property(Metacard.METADATA), "planning" ,
wildcardChar, singleChar, escapeChar, isCaseSensitive)
    );

DDF.catalog.operation.QueryImpl query = new QueryImpl( filter );
```

Tree View of Example Creating-Filters-3

Filters used in DDF can always be represented in a tree diagram.



XML View of Example Creating-Filters-3

Another way to view this type of Filter is through an XML model, which is shown below.

Pseudo XML of Example Creating-Filters-3

```
<Filter>
   <And>
      <PropertyIsLike wildCard="*" singleChar="?" escapeChar="\">
           <PropertyName>metadata</PropertyName>
           <Literal>mission</Literal>
      </PropertyIsLike>
      <PropertyIsLike wildCard="*" singleChar="?" escapeChar="\">
           <PropertyName>metadata</PropertyName>
           <Literal>planning</Literal>
      </PropertyIsLike>
   <And>
</Filter>
```

Using the Logical Operators and `PropertyIsLike` filters, a developer can create a whole language of search phrase expressions.

Fuzzy Operation

DDF only supports one custom function. The Filter specification does not include a fuzzy operator, so a Filter function was created to represent a fuzzy operation. The function and class is called `FuzzyFunction`, which is used by clients to notify the Sources to perform a fuzzy search. The syntax expected by providers is similar to the Fuzzy Function. Refer to the example below.

```
String wildcardChar = "*" ; // used to match zero or more characters
String singleChar = "?" ; // used to match exactly one character
String escapeChar = "\\" ; // used to escape the meaning of the wildCard, singleChar

boolean isCaseSensitive = false ;

Filter fuzzyFilter = filterFactory.like(
     new DDF.catalog.impl.filter.FuzzyFunction(
          Arrays.asList((Expression) (filterFactory.property(Metacard.ANY_TEXT))),
          filterFactory.literal("")),
     searchPhrase,
     wildcardChar,
     singleChar,
     escapeChar,
     isCaseSensitive);

QueryImpl query = new QueryImpl(fuzzyFilter);
```

Parsing Filters

According to the OGC Filter Specification 04-095: a "(filter expression) representation can be ... parsed

and then transformed into whatever target language is required to retrieve or modify object instances stored in some persistent object store." Filters can be thought of as the `WHERE` clause for a SQL SELECT statement to "fetch data stored in a SQL-based relational database."

Sources can parse OGC Filters using the `FilterAdapter` and `FilterDelegate`. See Developing a Filter Delegate for more details on implementing a new `FilterDelegate`. This is the preferred way to handle OGC Filters in a consistent manner.

Alternately, `org.opengis.filter.Filter` implementations can be parsed using implementations of the interface `org.opengis.filter.FilterVisitor`. The `FilterVisitor` uses the http://www.oodesign.com/visitor-pattern.html[Visitor pattern]. Essentially, `FilterVisitor` instances "visit" each part of the `Filter` tree allowing developers to implement logic to handle the filter's operations. Geotools 8 includes implementations of the `FilterVisitor` interface. The `DefaultFilterVisitor`, as an example, provides only business logic to visit every node in the `Filter` tree. The `DefaultFilterVisitor` methods are meant to be overwritten with the correct business logic. The simplest approach when using `FilterVisitor` instances is to build the appropriate query syntax for a target language as each part of the `Filter` is visited. For instance, when given an incoming `Filter` object to be evaluated against a RDBMS, a `CatalogProvider instance could use a 'FilterVisitor` to interpret each filter operation on the `Filter` object and translate those operations into SQL. The `FilterVisitor` may be needed to support `Filter` functionality not currently handled by the `FilterAdapter` and `FilterDelegate` reference implementation.

Examples

Interpreting a Filter to Create SQL

If the `FilterAdapter` encountered or "visited" a `PropertyIsLike` filter with its property assigned as `title` and its literal expression assigned as `mission`, the `FilterDelegate` could create the proper SQL syntax similar to title `LIKE` mission.

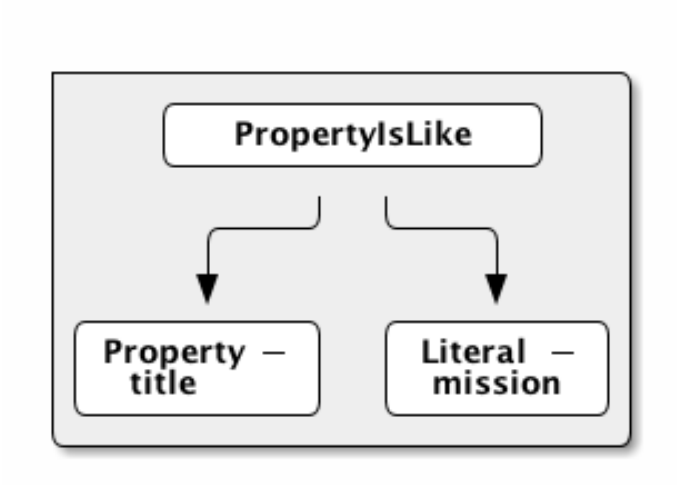


Figure 3. Figure Parsing-Filters1

Interpreting a Filter to Create XQuery

If the `FilterAdapter` encountered an `OR` filter, such as in Figure Parsing-Filters2 and the target language was XQuery, the `FilterDelegate` could yield an expression such as

```
ft:query(//inventory:book/@subject,'math') union
ft:query(//inventory:book/@subject,'science').
```

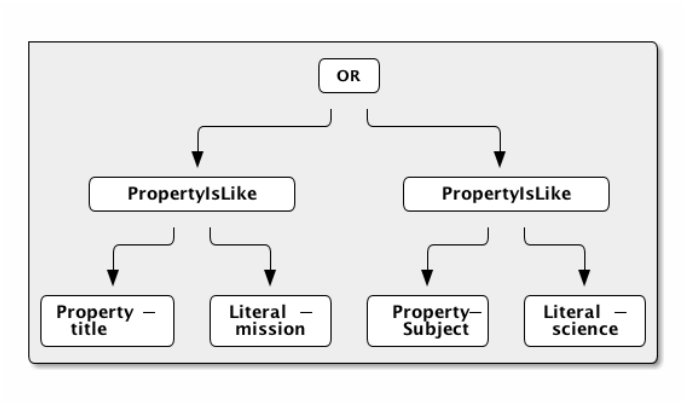


Figure 4. Figure Parsing-Filters2

FilterAdapter/Delegate Process for Figure Parsing-Filters2

1. `FilterAdapter` visits the `OR` filter first.
2. `OR` filter visits its children in a loop.
3. The first child in the loop that is encountered is the LHS `PropertyIsLike`.
4. The `FilterAdapter` will call the `FilterDelegate` `PropertyIsLike`method with the LHS property and literal.
5. The LHS `PropertyIsLike` delegate method builds the XQuery syntax that makes sense for this particular underlying object store. In this case, the *subject* property is specific to this XML database, and the business logic maps the *subject* property to its index at `//inventory:book/@subject` Note that `ft:query` in this instance is a custom XQuery module for this specific XML database that does full text searches.
6. The `FilterAdapter` then moves back to the `OR` filter, which visits its second child.
7. The `FilterAdapter` will call the `FilterDelegate PropertyIsLike` method with the RHS property and literal.
8. The RHS `PropertyIsLike` delegate method builds the XQuery syntax that makes sense for this particular underlying object store. In this case, the *subject* property is specific to this XML database, and the business logic maps the *subject* property to its index at `//inventory:book/@subject` Note that `ft:query` in this instance is a custom XQuery module for this specific XML database that does full text searches.

9. The `FilterAdapter` then moves back to its `OR Filter which is now done with its children.

10. It then collects the output of each child and sends the list of results to the `FilterDelegate OR` method.

11. The final result object will be returned from the `FilterAdapter` adapt method.

FilterVisitor Process for Figure Parsing-Filters2

1. FilterVisitor visits the `OR` filter first.
2. `OR` filter visits its children in a loop.
3. The first child in the loop that is encountered is the LHS `PropertyIsLike`.
4. The LHS `PropertyIsLike` builds the XQuery syntax that makes sense for this particular underlying object store. In this case, the *subject* property is specific to this XML database, and the business logic maps the *subject* property to its index at `//inventory:book/@subject`. Note that `ft:query` in this instance is a custom XQuery module for this specific XML database that does full text searches.
5. The FilterVisitor then moves back to the `OR` filter, which visits its second child.
6. The RHS `PropertyIsLike` builds the XQuery syntax that makes sense for this particular underlying object store. In this case, the *subject* property is specific to this XML database, and the business logic maps the *subject* property to its index at `//inventory:book/@subject`. Note that `ft:query` in this instance is a custom XQuery module for this specific XML database that does full text searches.
7. The FilterVisitor then moves back to its `OR` filter, which is now done with its children. It then collects the output of each child and could potentially execute the following code to produce the above expression.

```
public visit( Or filter, Object data) {
...
   /* the equivalent statement for the OR filter in this domain (XQuery) */
   xQuery = childFilter1Output + " union " + childFilter2Output;
...
}
```

4.5.4. Filter Profile

Role of the OGC Filter

Both Queries and Subscriptions extend the OGC GeoAPI Filter interface.

The Filter Builder and Adapter do not fully implement the OGC Filter Specification. The filter support profile contains suggested filter to metacard type mappings. For example, even though a Source could support a `PropertyIsGreaterThan` filter on `XML_TYPE`, it would not likely be useful.

Catalog Filter Profile

Metacard Attribute To Type Mapping

The filter profile maps filters to metacard types. The following table displays the common metacard attributes with their respective types for reference.

Metacard Attribute	Metacard Type
ANY_DATE	DATE_TYPE
ANY_GEO	GEO_TYPE
ANY_TEXT	STRING_TYPE
CONTENT_TYPE	STRING_TYPE
CONTENT_TYPE_VERSION	STRING_TYPE
CREATED	DATE_TYPE
EFFECTIVE	DATE_TYPE
GEOGRAPHY	GEO_TYPE
ID	STRING_TYPE
METADATA	XML_TYPE
MODIFIED	DATE_TYPE
RESOURCE_SIZE	STRING_TYPE
RESOURCE_URI	STRING_TYPE
SOURCE_ID	STRING_TYPE
TARGET_NAMESPACE	STRING_TYPE
THUMBNAIL	BINARY_TYPE
TITLE	STRING_TYPE

Comparison Operators

Comparison operators compare the value associated with a property name with a given Literal value. Endpoints and sources should try to use metacard types other than the object type. The object type only supports backwards compatibility with `java.net.URI`. Endpoints that send other objects will not be supported by standard sources. The following table maps the metacard types to supported comparison operators.

PropertyIs	Between	EqualTo	GreaterThan	GreaterThan	OrEqualTo	LessThan	LessThan	OrEqualTo	Like	NotEqualTo	Null
BINARY_TYPE		X									
BOOLEAN_TYPE		X									
DATE_TYPE	X	X	X	X	X	X	X	X		X	X
DOUBLE_TYPE	X	X	X	X	X	X	X	X		X	X
FLOAT_TYPE	X	X	X	X	X	X	X	X		X	X
GEO_TYPE											X
INTEGER_TYPE	X	X	X	X	X	X	X	X		X	X
LONG_TYPE	X	X	X	X	X	X	X	X		X	X
OBJECT_TYPE	X	X	X	X	X	X	X	X		X	X
SHORT_TYPE	X	X	X	X	X	X	X	X		X	X
STRING_TYPE	X	X	X	X	X	X	X	X	X	X	X
XML_TYPE		X							X		X

The following table describes each comparison operator.

Table 4. Comparison Operators

Operator	Description
PropertyIsBetween	Lower Property Upper

Operator	Description
PropertyIsEqualTo	Property == Literal
PropertyIsGreaterThan	Property > Literal
PropertyIsGreaterThanOrEqualTo	Property >= Literal
PropertyIsLessThan	Property < Literal
PropertyIsLessThanOrEqualTo	Property Literal
PropertyIsLike	Property LIKE Literal Equivalent to SQL "like"
PropertyIsNotEqualTo	Property != Literal
PropertyIsNull	Property == null

Logical Operators

Logical operators apply Boolean logic to one or more child filters.

Table 5. Logical Operators

	And	Not	Or
Supported Filters	**X**	**X**	**X**

Temporal Operators

Temporal operators compare a date associated with a property name to a given Literal date or date range. The following table displays the supported temporal operators.

	After	AnyInteracts	Before	Begins	BegunBy	During	EndedBy	Meets	MetBy	OverlappedBy	TContains
DATE_TYPE	**X**		**X**			**X**					

The following table describes each temporal operator. Literal values can be either date instants or date periods.

Operator	Description
After	Property > (Literal || Literal.end)
Before	Property < (Literal || Literal.start)
During	Literal.start < Property < Literal.end

Spatial Operators

Spatial operators compare a geometry associated with a property name to a given Literal geometry. The following table displays the supported spatial operators.

BBox	Beyond	Contains	Crosses	Disjoint	Equals	DWithin	Intersects	Overlaps	Touches	Within
GEO_TYPE		**X**	**X**	**X**	**X**		**X**	**X**	**X**	

The following table describes each spatial operator. Geometries are usually represented as Well-Known Text (*WKT*).

Operator	**Description**
Beyond	Property geometries beyond given distance of Literal geometry
Contains	Property geometry contains Literal geometry
Crosses	Property geometry crosses Literal geometry
Disjoint	Property geometry direct positions are not interior to Literal geometry
DWithin	Property geometry lies within distance to Literal geometry
Intersects	Property geometry intersects Literal geometry; opposite to the Disjoint operator
Overlaps	Property geometry interior somewhere overlaps Literal geometry interior
Touches	Property geometry touches but does not overlap Literal geometry
Within	Property geometry completely contains Literal geometry

4.5.5. Commons-DDF Utilities

The `commons-DDF`bundle, located in <DDF_HOME_SOURCE_DIRECTORY>/common/commons-DDF`, provides utilities and functionality commonly used across other DDF components, such as the endpoints and providers.

4.5.6. Noteworthy Classes

FuzzyFunction

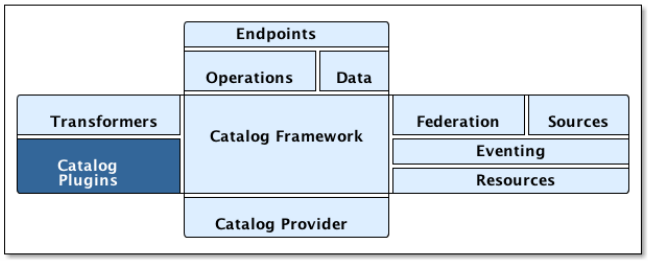
`DDF.catalog.impl.filter.FuzzyFunction` class is used to indicate that a `PropertyIsLike` filter should interpret the search as a fuzzy query.

XPathHelper

`DDF.util.XPathHelper` provides convenience methods for executing XPath operations on XML. It also provides convenience methods for converting XML as a `String` from a `org.w3c.dom.Document` object and vice versa.

4.6. Extending Catalog Plugins

The Catalog Framework calls Catalog Plugins to process requests and responses as they enter and leave the Framework.



4.6.1. Existing Plugins

Pre-Ingest Plugin

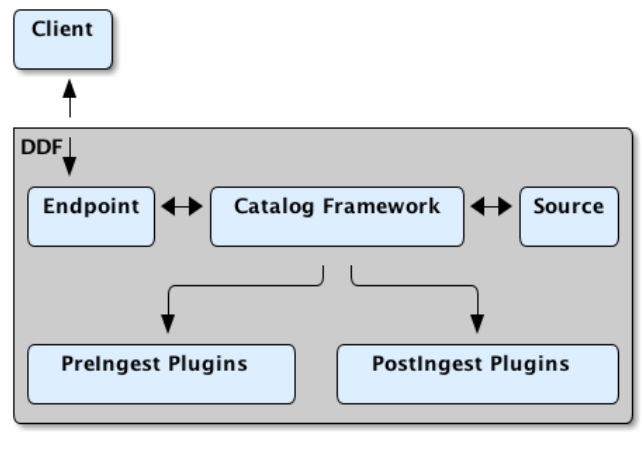


Figure 5. Ingest Plugin Flow

Using

Pre-Ingest plugins are invoked before an ingest operation is sent to a Source. This is an opportunity to

take any action on the ingest request, including but not limited to:

- validation
- logging
- auditing
- optimization
- security filtering

Failure Behavior

In the event that this Catalog Plugin cannot operate but does not wish to fail the transaction, a `PluginExecutionException` will be thrown. For any other Exceptions, the Catalog will "fail safe" and the Operation will be cancelled. If processing is to be explicitly stopped, a `StopProcessingException` will be thrown.

Invocation

Pre-Ingest plugins are invoked serially, prioritized by descending OSGi service ranking. The plugin with the highest service ranking will be executed first.

The output of a Pre-Ingest plugin is sent to the next Pre-Ingest plugin, until all have executed and the ingest operation is sent to the requested Source.

Metacard Groomer

The Metacard Groomer Pre-Ingest plugin makes modifications to `CreateRequest` and `UpdateRequest` metacards.

This plugin makes the following modifications when metacards are in a `CreateRequest`:

- Overwrites the `Metacard.ID` field with a generated, unique, 32 character hexadecimal value
- Overwrites the `Metacard.CREATED` date with a current time stamp
- Overwrites the `Metacard.MODIFIED` date with a current time stamp

The plugin also makes the following modifications when metacards are in an `UpdateRequest`:

- If no value is provided for `Metacard.ID` in the new metacard, it will be set using the `UpdateRequest` ID if applicable.
- If no value is provided, sets the `Metacard.CREATED` date with the `Metacard.MODIFIED` date so that the `Metacard.CREATED` date is not null.
- Overwrites the `Metacard.MODIFIED` date with a current time stamp.

Installing and UnInstalling

This plugin can be installed and uninstalled using the normal processes described in the Configuring DDF section.

Configuring

No configuration is necessary for this plugin.

Using

Use this pre-ingest plugin as a convenience to apply basic rules for your metacards.

Known Issues

None

Post-Ingest Plugin

Using

Post-ingest plugins are invoked after data has been created, updated, or deleted in a Catalog Provider.

Failure Behavior

In the event that this Catalog Plugin cannot operate but does not wish to fail the transaction, a `PluginExecutionException` will be thrown.

Invocation

Because the event has already occurred and changes from one post-ingest plugin cannot affect others, all Post-Ingest plugins are invoked in parallel and no priority is enforced.

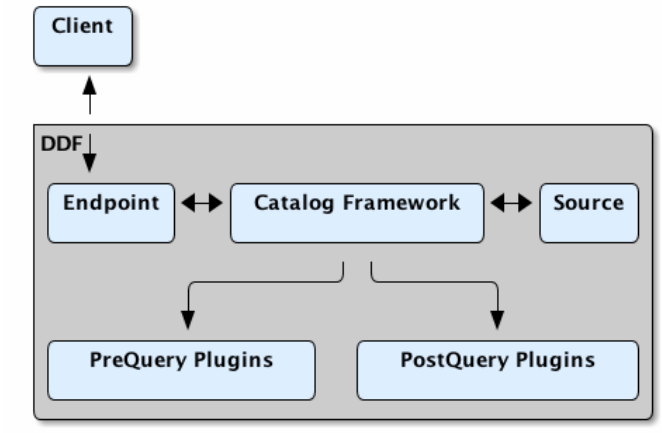


Figure 6. QueryPlugin Flow

Pre-Query Plugin

Using

Pre-query plugins are invoked before a query operation is sent to any of the Sources. This is an opportunity to take any action on the query, including but not limited to:

- validation
- logging
- auditing
- optimization
- security filtering

Failure Behavior

In the event that this Catalog Plugin cannot operate but does not wish to fail the transaction, a `PluginExecutionException` will be thrown. For any other Exceptions, the Catalog will "fail safe" and the Operation will be cancelled. If processing is to be explicitly stopped, a `StopProcessingException` will be thrown.

Invocation

Pre-query plugins are invoked serially, prioritized by descending OSGi service ranking. The plugin with the highest service ranking will be executed first. The output of a pre-query plugin is sent to the next pre-query plugin, until all have executed and the query operation is sent to the requested Source.

Post-Query Plugin

Using

Post-query plugins are invoked after a query has been executed successfully, but before the response is returned to the endpoint. This is an opportunity to take any action on the query response, including but not limited to:

- logging
- auditing
- security filtering/redaction
- deduplication

Failure Behavior

In the event that this Catalog Plugin cannot operate but does not wish to fail the transaction, a `PluginExecutionException` will be thrown. For any other Exceptions, the Catalog will "fail safe" and the

Operation will be cancelled. If processing is to be explicitly stopped, a `StopProcessingException` will be thrown.

Invocation

Post-query plugins are invoked serially, prioritized by descending OSGi service ranking. The plugin with the highest service ranking will be executed first. The output of the first plugin is sent to the next plugin, until all have executed and the response is returned to the requesting endpoint.

4.6.2. Metacard Resource Size Plugin

This post-query plugin updates the resource size attribute of each metacard in the query results if there is a cached file for the product and it has a size greater than zero; otherwise, the resource size is unmodified and the original result is returned.

Installing and UnInstalling

This feature can be installed and uninstalled using the normal processes described in the Configuring DDF section.

Configuring

No configuration is necessary for this plugin.

Using

Use this post-query plugin as a convenience to return query results with accurate resource sizes for cached products.

Known Issues

None

4.6.3. Other Types of Plugins

Pre-Get Resource Plugin

Using

Pre-get resource plugins are invoked before a request to retrieve a resource is sent to a Source. This is an opportunity to take any action on the request, including but not limited to:

- validation
- logging
- auditing
- optimization

- security filtering

Failure Behavior

In the event that this Catalog Plugin cannot operate but does not wish to fail the transaction, a `PluginExecutionException` will be thrown. For any other Exceptions, the Catalog will "fail safe" and the Operation will be cancelled. If processing is to be explicitly stopped, a `StopProcessingException` will be thrown.

Invocation

Pre-get resource plugins are invoked serially, prioritized by descending OSGi service ranking. That is, the plugin with the highest service ranking will be executed first.

The output of the first plugin is sent to the next plugin, until all have executed and the request is sent to the targeted Source.

Post-Get Resource Plugin

Using

Post-get resource plugins are invoked after a resource has been retrieved, but before it is returned to the endpoint. This is an opportunity to take any action on the response, including but not limited to:

- logging
- auditing
- security filtering/redaction

Failure Behavior

In the event that this Catalog Plugin cannot operate but does not wish to fail the transaction, a `PluginExecutionException` will be thrown. For any other Exceptions, the Catalog will "fail safe" and the Operation will be cancelled. If processing is to be explicitly stopped, a `StopProcessingException` will be thrown.

Invocation

Post-get resource plugins are invoked serially, prioritized by descending OSGi service ranking. The plugin with the highest service ranking will be executed first.

The output of the first plugin is sent to the next plugin, until all have executed and the response is returned to the requesting endpoint.

Pre-Subscription Plugin

Using

Pre-subscription plugins are invoked before a Subscription is activated by an Event Processor. This is an opportunity to take any action on the Subscription, including but not limited to:

- validation
- logging
- auditing
- optimization
- security filtering

Failure Behavior

In the event that this Catalog Plugin cannot operate but does not wish to fail the transaction, a `PluginExecutionException` will be thrown. For any other Exceptions, the Catalog will "fail safe" and the Operation will be cancelled. If processing is to be explicitly stopped, a `StopProcessingException` will be thrown.

Invocation

Pre-subscription plugins are invoked serially, prioritized by descending OSGi service ranking. That is, the plugin with the highest service ranking will be executed first.

The output of a pre-subscription plugin is sent to the next pre-subscription plugin, until all have executed and the create Subscription operation is sent to the Event Processor.

Examples

DDF includes a pre-subscription plugin example in the SDK that illustrates how to modify a subscription's filter. This example is located in the DDF trunk at `sdk/sample-plugins/DDF/sdk/plugin/presubscription`.

Pre-Delivery Plugin

Using

Pre-delivery plugins are invoked before a Delivery Method is invoked on a Subscription. This is an opportunity to take any action before notification, including but not limited to:

- logging
- auditing
- security filtering/redaction

Failure Behavior

In the event that this Catalog Plugin cannot operate but does not wish to fail the transaction, a `PluginExecutionException` will be thrown. For any other Exceptions, the Catalog will "fail safe" and the Operation will be cancelled. If processing is to be explicitly stopped, a `StopProcessingException` will be thrown.

Invocation

Pre-delivery plugins are invoked serially, prioritized by descending OSGi service ranking. The plugin with the highest service ranking will be executed first.

The output of a pre-delivery plugin is sent to the next pre-delivery plugin, until all have executed and the Delivery Method is invoked on the associated Subscription.

4.6.4. Developing a Catalog Plugin

4.6.5. Policy Plugin

Using

Policy plugins are invoked before all other plugin types to set up the policy for a request/response. This provides an opportunity to attach custom requirements on operations or individual metacards. All the 'requirements' from each Policy plugin will be combined into a single policy that will be included in the request/response. Access plugins will be used to act on this combined policy.

Failure Behavior

All failure cases should be handled internally to the plugin with the exception of the StopProcessingException. If the exception encountered should stop/block the request then a StopProcessingException should be thrown.

4.6.6. Access Plugin

Using

Access plugins are invoked directly after the Policy plugins have been successfully executed. This is an opportunity to either stop processing or modify the request/response based on policy information.

Failure Behavior

All failure cases should be handled internally to the plugin with the exception of the StopProcessingException. If the exception encountered should stop/block the request then a StopProcessingException should be thrown.

4.6.7. Developing a Catalog Plugin

Plugins extend the functionality of the Catalog Framework by performing actions at specified times

during a transaction. Plugins can be *Pre-Ingest*, *Post-Ingest*, *Pre-Query*, *Post-Query*, *Pre-Subscription*, *Pre-Delivery*, *Pre-Resource*, or *Post-Resource*. By implementing these interfaces, actions can be performed at the desired time.

Create New Plugins

Implement Plugin Interface

The following types of plugins can be created:

Plugin Type	Plugin Interface	Description	Example
Pre-Ingest	`DDF.catalog.plugin.PreIngestPlugin`	Runs before the Create/Update/Delete method is sent to the CatalogProvider	Metadata validation services
Post-Ingest	`DDF.catalog.plugin.PostIngestPlugin`	Runs after the Create/Update/Delete method is sent to the CatalogProvider	EventProcessor for processing and publishing event notifications to subscribers
Pre-Query	`DDF.catalog.plugin.PreQueryPlugin`	Runs prior to the Query/Read method being sent to the Source	An example is not included with DDF
Post-Query	`DDF.catalog.plugin.PostQueryPlugin`	Runs after results have been retrieved from the query but before they are posted to the Endpoint	An example is not included with DDF
Pre-Subscription	`DDF.catalog.plugin.PreSubscription`	Runs prior to a Subscription being created or updated	Modify a query prior to creating a subscription
Pre-Delivery	`DDF.catalog.plugin.PreDeliveryPlugin`	Runs prior to the delivery of a Metacard when an event is posted	Inspect a metacard prior to delivering it to the Event Consumer
Pre-Resource	`DDF.catalog.plugin.PreResource`	Runs prior to a Resource being retrieved	An example is not included with DDF
Post-Resource	`DDF.catalog.plugin.PostResource`	Runs after a Resource is retrieved, but before it is sent to the Endpoint	Verification of a resource prior to returning to a client
Policy	DDF.catalog.plugin.PolicyPlugin	Runs prior to all other catalog plugins to esablish the policy for requests/responses	An example is MetacardValidityFilterPlugin

Plugin Type	Plugin Interface	Description	Example
Access	DDF.catalog.plugin.AccessPlugin	Runs directly after the PolicyPlugin	An examples are the FilterPlugin and OperationPlugin

Implement Plugins

The procedure for implementing any of the plugins follows a similar format:

1. Create a new class that implements the specified plugin interface.
2. Implement the required methods.
3. Create OSGi descriptor file to communicate with the OSGi registry.
   a. Import DDF packages.
   b. Register plugin class as service to OSGi registry.
4. Deploy to DDF.

> **TIP** Refer to the Javadoc for more information on all Requests and Responses in the `ddf.catalog.operation` and `ddf.catalog.event` packages.

Pre-Ingest

1. Create a Java class that implements `PreIngestPlugin.`
   `public class SamplePreIngestPlugin implements DDF.catalog.plugin.PreIngestPlugin`
2. Implement the required methods.
   - `public CreateRequest process(CreateRequest input) throws PluginExecutionException;`
   - `public UpdateRequest process(UpdateRequest input) throws PluginExecutionException;`
   - `public DeleteRequest process(DeleteRequest input) throws PluginExecutionException;`
3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: DDF.catalog,DDF.catalog.plugin`
4. Export the service to the OSGi registry.
   Blueprint descriptor example `<service ref="[[SamplePreIngestPlugin ]]"interface="DDF.catalog.plugin.PreIngestPlugin" />`

Post-Ingest

1. Create a Java class that implements `PostIngestPlugin`.
   `public class SamplePostIngestPlugin implements DDF.catalog.plugin.PostIngestPlugin`

2. Implement the required methods.
   - `public CreateResponse process(CreateResponse input) throws PluginExecutionException;`
   - `public UpdateResponse process(UpdateResponse input) throws PluginExecutionException;`
   - `public DeleteResponse process(DeleteResponse input) throws PluginExecutionException;`
3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: DDF.catalog,DDF.catalog.plugin`
4. Export the service to the OSGi registry.
   Blueprint descriptor example `<service ref="[[SamplePostIngestPlugin ]]"interface="DDF.catalog.plugin.PostIngestPlugin" />`

Pre-Query

1. Create a Java class that implements `PreQueryPlugin`.
   `public class SamplePreQueryPlugin implements DDF.catalog.plugin.PreQueryPlugin`
2. Implement the required method.
   `public QueryRequest process(QueryRequest input) throws PluginExecutionException, StopProcessingException;`
3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: DDF.catalog,DDF.catalog.plugin`
4. Export the service to the OSGi registry.
   `<service ref=""interface="DDF.catalog.plugin.PreQueryPlugin" />`

Post-Query

1. Create a Java class that implements `PostQueryPlugin`.
   `public class SamplePostQueryPlugin implements DDF.catalog.plugin.PostQueryPlugin`
2. Implement the required method.
   `public QueryResponse process(QueryResponse input) throws PluginExecutionException, StopProcessingException;`
3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: DDF.catalog,DDF.catalog.plugin`
4. Export the service to the OSGi registry.
   `<service ref=""interface="DDF.catalog.plugin.PostQueryPlugin" />`

Pre-Delivery

1. Create a Java class that implements `PreDeliveryPlugin`.
   `public class SamplePreDeliveryPlugin implements DDF.catalog.plugin.PreDeliveryPlugin`

2. Implement the required methods.
   `public Metacard processCreate(Metacard metacard) throws PluginExecutionException, StopProcessingException; public Update processUpdateMiss(Update update) throws PluginExecutionException, StopProcessingException;`

   - `public Update processUpdateHit(Update update) throws PluginExecutionException, StopProcessingException;`
   - `public Metacard processCreate(Metacard metacard) throws PluginExecutionException, StopProcessingException;`

3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: DDF.catalog,DDF.catalog.plugin,DDF.catalog.operation,DDF.catalog.event`

4. Export the service to the OSGi registry.
   Blueprint descriptor example
   `<service ref=""interface="DDF.catalog.plugin.PreDeliveryPlugin" />`

Pre-Subscription

1. Create a Java class that implements `PreSubscriptionPlugin`.
   `public class SamplePreSubscriptionPlugin **implements** DDF.catalog.plugin.PreSubscriptionPlugin

2. Implement the required method.

   - `public Subscription process(Subscription input) throws PluginExecutionException, StopProcessingException;`

Pre-Resource

1. Create a Java class that implements `PreResourcePlugin`. `public class SamplePreResourcePlugin implements DDF.catalog.plugin.PreResourcePlugin`

2. Implement the required method.

   - `public ResourceRequest process(ResourceRequest input) throws PluginExecutionException, StopProcessingException;`

3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: DDF.catalog,DDF.catalog.plugin,DDF.catalog.operation`

4. Export the service to the OSGi registry. .Blueprint descriptor example

```
<service ref="[[SamplePreResourcePlugin]]"
interface="DDF.catalog.plugin.PreResourcePlugin" />
```

Post-Resource

1. Create a Java class that implements `PostResourcePlugin`.
   `public class SamplePostResourcePlugin implements DDF.catalog.plugin.PostResourcePlugin`
2. Implement the required method.
   - `public ResourceResponse process(ResourceResponse input) throws PluginExecutionException, StopProcessingException;`
3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: DDF.catalog,DDF.catalog.plugin,DDF.catalog.operation`
4. Export the service to the OSGi registry.

Blueprint descriptor example

```
<service ref="[[SamplePostResourcePlugin]]" interface=
"DDF.catalog.plugin.PostResourcePlugin" />
```

Policy

1. Create a Java class that implements `PolicyPlugin`.
   `public class SamplePolicyPlugin implements DDF.catalog.plugin.PolicyPlugin`
2. Implement the required methods.
   - `PolicyResponse processPreCreate(Metacard input, Map<String, Serializable> properties) throws StopProcessingException;`
   - `PolicyResponse processPreUpdate(Metacard input, Map<String, Serializable> properties) throws StopProcessingException;`
   - `PolicyResponse processPreDelete(String attributeName, List<Serializable> attributeValues, Map<String, Serializable> properties) throws StopProcessingException;`
   - `PolicyResponse processPreQuery(Query query, Map<String, Serializable> properties) throws StopProcessingException;`
   - `PolicyResponse processPostQuery(Result input, Map<String, Serializable> properties) throws StopProcessingException;`
3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: DDF.catalog,DDF.catalog.plugin,DDF.catalog.operation`

4. Export the service to the OSGi registry.
   Blueprint descriptor example
   ```
   <service ref="" interface="DDF.catalog.plugin.PolicyPlugin" />
   ```

Access

1. Create a Java class that implements `AccessPlugin`.
   ```
   public class SamplePostResourcePlugin implements DDF.catalog.plugin.AccessPlugin
   ```

2. Implement the required methods.

   - `CreateRequest processPreCreate(CreateRequest input) throws StopProcessingException;`
   - `UpdateRequest processPreUpdate(UpdateRequest input) throws StopProcessingException;`
   - `DeleteRequest processPreDelete(DeleteRequest input) throws StopProcessingException;`
   - `QueryRequest processPreQuery(QueryRequest input) throws StopProcessingException;`
   - `QueryResponse processPostQuery(QueryResponse input) throws StopProcessingException;`

3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   ```
   Import-Package: DDF.catalog,DDF.catalog.plugin,DDF.catalog.operation
   ```

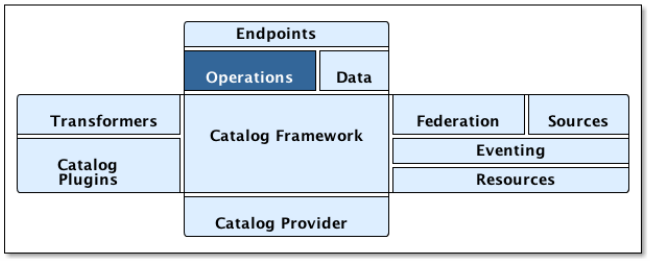
4. Export the service to the OSGi registry.
   Blueprint descriptor example
   ```
   <service ref="" interface="DDF.catalog.plugin.AccessPlugin" />
   ```

4.7. Extending Operations

The Catalog provides the capability to query, create, update, and delete metacards; retrieve resources; and retrieve information about the sources in the enterprise.

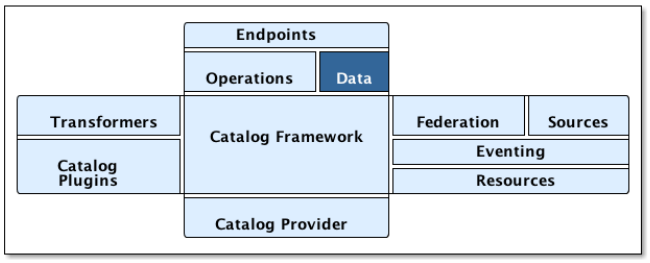
Each of these operations follow a request/response paradigm. The request is the input to the operation and contains all of the input parameters needed by the Catalog Framework's operation to communicate with the Sources. The response is the output from the execution of the operation that is returned to the client, which contains all of the data returned by the sources. For each operation there is an associated request/response pair, e.g., the `QueryRequest` and `QueryResponse` pair for the Catalog Framework's query operation.

All of the request and response objects are extensible in that they can contain additional key/value properties on each request/response. This allows additional capability to be added without changing the Catalog API, helping to maintain backwards compatibility.



4.7.1. Extending Data and Metadata Basics

The catalog stores and translates Metadata which can be transformed into many data formats, shared, and queried. The primary form of this metadata is the metacard. A `Metacard` is a container for metadata. `CatalogProviders` accept `Metacards` as input for ingest, and `Sources` search for metadata and return matching `Results` that include `Metacards`.



4.7.2. Metacard

A single instance of metadata in the Catalog (an instance of a metacard type) which generally contains metadata providing a title for the product and describing a product's geo-location, created and modified dates, owner or producer, security classification, etc.

4.7.3. Metacard Type

Metacard Type

A metacard type indicates the attributes available for a particular metacard. It is a model used to define the attributes of a metacard, much like a schema.

Default Metacard Type and Attributes

Most metacards within the system are created using with the default metacard type. The default metacard type of the system can be programmatically retrieved by calling `DDF.catalog.data.BasicTypes.BASIC_METACARD`. The name of the default MetacardType can be retrieved from `DDF.catalog.data.MetacardType.DEFAULT_METACARD_TYPE_NAME`.

The default metacard type has the following required attributes. Though the following attributes are required on all metacard types, setting their values is optional except for ID.

Required Attributes

`DDF.catalog.data.Metacard` Constant	Attribute Name	Attribute Format	Description
`CONTENT_TYPE`	`metadata-content-type`	STRING	Attribute name accessing for the metadata content type of a Metacard.
`CONTENT_TYPE_VERSION`	`metadata-content-type-version`	STRING	Attribute name for the version of the metadata content accessing type of a Metacard.
`CREATED`	`created`	DATE	Attribute name for accessing the date/time **this Metacard** was created.
`EFFECTIVE`	`effective`	DATE	Attribute name for accessing the date/time of **the product** represented by the Metacard.
`EXPIRATION`	`expiration`	DATE	Attribute name for accessing the date/time the Metacard is no longer valid and could be removed.
`GEOGRAPHY`	`location`	GEOMETRY	Attribute name for accessing the location for this Metacard.
`ID`	`id`	STRING	Attribute name for accessing the ID of the Metacard.
`METADATA`	`metadata`	XML	Attribute name for accessing the XML metadata for this Metacard.
`MODIFIED`	`modified`	DATE	Attribute name for accessing the date/time this Metacard was last modified.

DDF.catalog.data.Metacard Constant	Attribute Name	Attribute Format	Description
RESOURCE_SIZE	resource-size	STRING	Attribute name for accessing the size in bytes of the product this Metacard represents.
RESOURCE_URI	resource-uri	STRING	Attribute name for accessing the URI reference to the product this Metacard represents.
TARGET_NAMESPACE	metadata-target-namespace	STRING	Attribute name for the target namespace of the accessing metadata content type of a Metacard.
THUMBNAIL	thumbnail	BINARY	Attribute name for accessing the thumbnail image of the product this Metacard represents. The thumbnail must be of MIME Type image/jpeg and be less than 128 kilobytes.
TITLE	title	STRING	Attribute name for accessing the title of the Metacard.

NOTE It is highly recommended when referencing a default attribute name to use the DDF.catalog.data.Metacard constants whenever possible.

WARNING Every Source should at the very least return an ID attribute according to Catalog API. Other fields might or might not be applicable, but a unique ID must be returned by a Source.

Extensible Metacards

Metacard extensibility is achieved by creating a new MetacardType that supports attributes in addition to the required attributes listed above.

Required attributes must be the base of all extensible metacard types.

WARNING	Not all Catalog Providers support extensible metacards. Nevertheless, each Catalog Provider should at least have support for the default `MetacardType`; i.e., it should be able to store and query on the attributes and attribute formats specified by the default metacard type. Consult the documentation of the Catalog Provider in use for more information on its support of extensible metacards.

Metacard Extensibility

Often, the `BASIC_METACARD` `MetacardType` does not provide all the functionality or attributes necessary for a specific task. For performance or convenience purposes, it may be necessary to create custom attributes even if others will not be aware of those attributes. One example could be if a user wanted to optimize a search for a date field that did not fit the definition of `CREATED`, `MODIFIED`, `EXPIRATION`, or `EFFECTIVE`. The user could create an additional `java.util.Date` attribute in order to query the attribute separately.

`Metacard` objects are extensible because they allow clients to store and retrieve standard and custom key/value Attributes from the `Metacard`. All `Metacards` must return a `MetacardType` object that includes an `AttributeDescriptor` for each `Attribute`, indicating it's key and value type. `AttributeType` support is limited to those types defined by the Catalog.

New `MetacardType` implementations can be made by implementing the `MetacardType` interface.

4.7.4. Metacard Type Registry

WARNING	The `MetacardTypeRegistry` is experimental. While this component has been tested and is functional, it may change as more information is gathered about what is needed and as it is used in more scenarios.

The `MetacardTypeRegistry` allows DDF components, primarily CatalogProviders and Sources, to make available the `MetacardTypes` that they support. It maintains a list of all supported `MetacardTypes` in the `CatalogFramework`, so that other components such as Endpoints, Plugins, and Transformers can make use of those `MetacardTypes`. The `MetacardType` is essential for a component in the `CatalogFramework` to understand how it should interpret a metacard by knowing what attributes are available in that metacard.

For example, an endpoint receiving incoming metadata can perform a lookup in the `MetacardTypeRegistry` to find a corresponding `MetacardType`. The discovered `MetacardType` will then be used to help the endpoint populate a metacard based on the specified attributes in the `MetacardType`. By doing this, all the incoming metadata elements can then be available for processing, cataloging, and searching by the rest of the `CatalogFramework`.

`MetacardTypes` should be registered with the `MetacardTypeRegistry`. The `MetacardTypeRegistry` makes those `MetacardTypes` available to other DDF `CatalogFramework` components. Other components that need to know how to interpret metadata or metacards should look up the appropriate `MetacardType` from the registry. By having these `MetacardTypes` available to the `CatalogFramework`, these components can be aware of the custom attributes.

The `MetacardTypeRegistry` is accessible as an OSGi service. The following blueprint snippet shows how to inject that service into another component:

```
<bean id="sampleComponent" class="DDF.catalog.SampleComponent">
    <argument ref="metacardTypeRegistry" />
</bean>

<!-- Access MetacardTypeRegistry -->
<reference id="metacardTypeRegistry" interface="DDF.catalog.data.MetacardTypeRegistry"/>
```

The reference to this service can then be used to register new `MetacardTypes` or to lookup existing ones.

Typically, new `MetacardTypes` will be registered by `CatalogProviders` or Sources indicating they know how to persist, index, and query attributes from that type. Typically, Endpoints or `InputTransformers` will use the lookup functionality to access a `MetacardType` based on a parameter in the incoming metadata. Once the appropriate `MetacardType` is discovered and obtained from the registry, the component will know how to translate incoming raw metadata into a DDF Metacard.

Attribute

A single field of a metacard, an instance of an attribute type. Attributes are typically indexed for searching by a Source or Catalog Provider.

Attribute Type

An attribute type indicates the attribute format of the value stored as an attribute. It is a model for an attribute.

Attribute Format

An enumeration of attribute formats are available in the catalog. Only these attribute formats may be used.

AttributeFormat	Description
`BINARY`	Attributes of this attribute format must have a value that is a Java `byte[]` and `AttributeType.getBinding()` should return `Class<Array>of` byte.
`BOOLEAN`	Attributes of this attribute format must have a value that is a Java boolean.
`DATE`	Attributes of this attribute format must have a value that is a Java date.
`DOUBLE`	Attributes of this attribute format must have a value that is a Java double.

AttributeFormat	Description
`FLOAT`	Attributes of this attribute format must have a value that is a Java float.
`GEOMETRY`	Attributes of this attribute format must have a value that is a WKT-formatted Java string.
`INTEGER`	Attributes of this attribute format must have a value that is a Java integer.
`LONG`	Attributes of this attribute format must have a value that is a Java long.
`OBJECT`	Attributes of this attribute format must have a value that implements the serializable interface.
`SHORT`	Attributes of this attribute format must have a value that is a Java short.
`STRING`	Attributes of this attribute format must have a value that is a Java string and treated as plain text.
`XML`	Attributes of this attribute format must have a value that is a XML-formatted Java string.

Result

A single "hit" included in a query response.

A result object consists of the following:

- a metacard
- a relevance score if included
- distance in meters if included

Creating Metacards

The quickest way to create a `Metacard` is to extend or construct the `MetacardImpl` object. `MetacardImpl` is the most commonly used and extended `Metacard` implementation in the system because it provides a convenient way for developers to retrieve and set `Attribute`s without having to create a new `MetacardType` (see below). `MetacardImpl` uses `BASIC_METACARD` as its `MetacardType`.

Limitations

A given developer does not have all the information necessary to programmatically interact with any arbitrary `Source`. Developers hoping to query custom fields from extensible `Metacards` of other `Sources` cannot easily accomplish that task with the current API. A developer cannot question a random `Source` for all its *queryable* fields. A developer only knows about the `MetacardTypes` which that

individual developer has used or created previously.

The only exception to this limitation is the `Metacard.ID` field, which is required in every `Metacard` that is stored in a `Source`. A developer can always request `Metacards` from a `Source` for which that developer has the `Metacard.ID` value. The developer could also perform a wildcard search on the `Metacard.ID` field if the `Source` allows.

Processing Metacards

As `Metacard` objects are created, updated, and read throughout the Catalog, care should be taken by all Catalog Components to interrogate the `MetacardType` to ensure that additional `Attributes` are processed accordingly.

Basic Types

The Catalog includes definitions of several Basic Types all found in the `DDF.catalog.data.BasicTypes` class.

Name	Type	Description
`BASIC_METACARD`	MetacardType	representing all required Metacard Attributes
`BINARY_TYPE`	AttributeType	A Constant for an AttributeType with `AttributeType.AttributeFormat.BINARY`.
`BOOLEAN_TYPE`	AttributeType	A Constant for an AttributeType with `AttributeType.AttributeFormat.BOOLEAN`.
`DATE_TYPE`	AttributeType	A Constant for an AttributeType with `AttributeType.AttributeFormat.DATE`.
`DOUBLE_TYPE`	AttributeType	A Constant for an AttributeType with `AttributeType.AttributeFormat.DOUBLE`.
`FLOAT_TYPE`	AttributeType	A Constant for an AttributeType with `AttributeType.AttributeFormat.FLOAT`.
`GEO_TYPE`	AttributeType	A Constant for an AttributeType with `AttributeType.AttributeFormat.GEOMETRY`.
`INTEGER_TYPE`	AttributeType	A Constant for an AttributeType with `AttributeType.AttributeFormat.INTEGER`.

Name	Type	Description
LONG_TYPE	AttributeType	A Constant for an AttributeType with AttributeType.AttributeFormat.LONG.
OBJECT_TYPE	AttributeType	A Constant for an AttributeType with AttributeType.AttributeFormat.OBJECT.
SHORT_TYPE	AttributeType	A Constant for an AttributeType with AttributeType.AttributeFormat.SHORT.
STRING_TYPE	AttributeType	A Constant for an AttributeType with AttributeType.AttributeFormat.STRING.
XML_TYPE	AttributeType	A Constant for an AttributeType with AttributeType.AttributeFormat.XML.

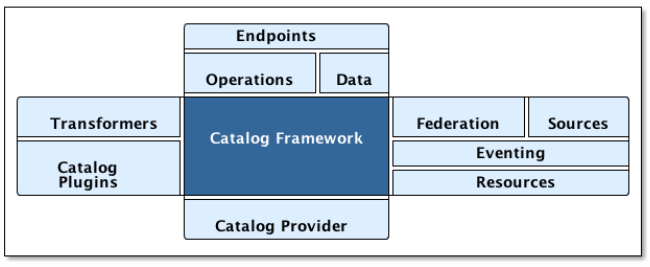
4.8. Extending Catalog Framework

This section describes the core components of the Catalog app and Catalog Framework. The Catalog Framework wires all Catalog components together.

It is responsible for routing Catalog requests and responses to the appropriate target.

Endpoints send Catalog requests to the Catalog Framework. The Catalog Framework then invokes Catalog Plugins, Transformers, and Resource Components as needed before sending requests to the intended destination, such as one or more Sources.

The Catalog Framework functions as the routing mechanisms between all catalog components. It decouples clients from service implementations and provides integration points for Catalog Plugins and convenience methods for Endpoint developers.



4.8.1. Included Catalog Frameworks

Catalog API

The Catalog API is an OSGi bundle (`catalog-core-api`) that contains the Java interfaces for the Catalog components and implementation classes for the Catalog Framework, Operations, and Data components.

Standard Catalog Framework

The Standard Catalog Framework provides the reference implementation of a Catalog Framework that implements all requirements of the DDF CatalogAPI. `CatalogFrameworkImpl` is the implementation of the DDF Standard Catalog Framework.

Installing and Uninstalling

The Standard Catalog Framework is bundled as the `catalog-core-standardframework` feature and can be installed and uninstalled using the normal processes described in Configuration.

When this feature is installed, the Catalog Fanout Framework App feature `catalog-core-fanoutframework` should be uninstalled, as both catalog frameworks should not be installed simultaneously.

Configuring

Configurable Properties

Table 6. Catalog Standard Framework

Property	Type	Description	Default Value	Required
`fanoutEnabled`	Boolean	When enabled the Framework acts as a proxy, federating requests to all available sources. All requests are executed as federated queries and resource retrievals, allowing the framework to be the sole component exposing the functionality of all of its Federated Sources.	false	yes

Property	Type	Description	Default Value	Required
`productCacheDirectory`	String	Directory where retrieved products will be cached for faster, future retrieval. If a directory path is specified with directories that do not exist, Catalog Framework will attempt to create those directories. Out of the box (without configuration), the product cache directory is `<INSTALL_DIR>/data/product-cache`. If a relative path is provided it will be relative to the `<INSTALL_DIR>`. It is recommended to enter an absolute directory path such as `/opt/product-cache`in Linux or `C:\product-cache` in Windows."	`(empty)`	no
`cacheEnabled`	Boolean	Check to enable caching of retrieved products to provide faster retrieval for subsequent requests for the same product.	false	no
`delayBetweenRetryAttempts`	Integer	The time to wait (in seconds) between each attempt to retry retrieving a product from the Source.	10	no
`maxRetryAttempts`	Integer	The maximum number of attempts to try and retrieve a product from the Source.	3	no
`cachingMonitorPeriod`	Integer	The number of seconds allowed for no data to be read from the product data before determining that the network connection to the Source where the product is located is considered to be down.	5	no
`cacheWhenCanceled`	Boolean	Check to enable caching of retrieved products even if client cancels the download.	false	no

Managed Service PID	`DDF.catalog.CatalogFrameworkImpl`
Managed Service Factory PID	N/A

Using

The Standard Catalog Framework is the core class of DDF. It provides the methods for query, create, update, delete, and resource retrieval (QCRUD) operations on the `Sources`. By contrast, the Fanout Catalog Framework only allows for query and resource retrieval operations, no catalog modifications, and all queries are enterprise-wide.

Use this framework if:

- access to a catalog provider to create, update, and delete catalog entries are required.
- queries to specific sites are required.
- queries to only the local provider are required.

It is possible to have only remote Sources configured with no local `CatalogProvider` configured and be able to execute queries to specific remote sources by specifying the site name(s) in the query request.

The Standard Catalog Framework also maintains a list of `ResourceReaders` for resource retrieval operations. A resource reader is matched to the scheme (i.e., protocol, such as `file://`) in the URI of the resource specified in the request to be retrieved.

Site information about the catalog provider and/or any federated source(s) can be retrieved using the Standard Catalog Framework. Site information includes the source's name, version, availability, and the list of unique content types currently stored in the source (e.g., NITF). If no local catalog provider is configured, the site information returned includes site info for the catalog framework with no content types included.

Implementation Details

Exported Services

Registered Interface	Service Property	Value
`DDF.catalog.federation.FederationStrategy`	shortname	sorted
`org.osgi.service.event.EventHandler`	event.topics	`DDF/catalog/event/CREATED`, `DDF/catalog/event/UPDATED`, d`df/catalog/event/DELETED`
`DDF.catalog.CatalogFramework`		
`DDF.catalog.event.EventProcessor`		
`DDF.catalog.plugin.PostIngestPlugin`		

Imported Services

Registered Interface	Availability	Multiple
DDF.catalog.plugin.PostFederatedQueryPlugin	optional	true
DDF.catalog.plugin.PostIngestPlugin	optional	true
DDF.catalog.plugin.PostQueryPlugin	optional	true
DDF.catalog.plugin.PostResourcePlugin	optional	true
DDF.catalog.plugin.PreDeliveryPlugin	optional	true
DDF.catalog.plugin.PreFederatedQueryPlugin	optional	true
DDF.catalog.plugin.PreIngestPlugin	optional	true
DDF.catalog.plugin.PreQueryPlugin	optional	true
DDF.catalog.plugin.PreResourcePlugin	optional	true
DDF.catalog.plugin.PreSubscriptionPlugin	optional	true
DDF.catalog.plugin.PolicyPlugin	optional	true
DDF.catalog.plugin.AccessPlugin	optional	true
DDF.catalog.resource.ResourceReader	optional	true
DDF.catalog.source.CatalogProvider	optional	true
DDF.catalog.source.ConnectedSource	optional	true
DDF.catalog.source.FederatedSource	optional	true
DDF.cache.CacheManager		false
org.osgi.service.event.EventAdmin		false

4.8.2. Known Issues

None

4.9. Catalog Fanout Framework

The Fanout Catalog Framework (`fanout-catalogframework` bundle) provides an implementation of the Catalog Framework that acts as a proxy, federating requests to all available sources. All requests are executed as federated queries and resource retrievals, allowing the fanout site to be the sole site exposing the functionality of all of its Federated Sources. The Fanout Catalog Framework is the implementation of the Fanout Catalog Framework.

The Fanout Catalog Framework provides the capability to configure DDF to be a fanout proxy to other federated sources within the enterprise. The Fanout Catalog Framework has no catalog provider configured for it, so it does not allow catalog modifications to take place. Therefore, create, update, and delete operations are not supported.

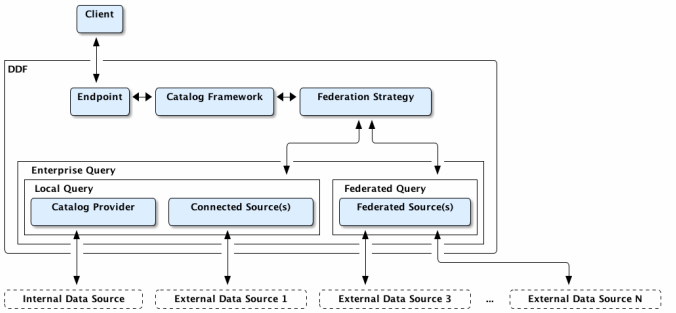


Figure 7. Catalog Fanout Framework

In addition, the Fanout Catalog Framework provides the following benefits:

- Backwards compatibility (e.g., federating with older versions) with existing older versions of DDF.
- A single node being exposed from an enterprise, thus hiding the enterprise from an external client.
- Ensures all queries and resource retrievals are federated.

Installing and Uninstalling

The Fanout Catalog Framework is bundled as the `catalog-core-fanoutframework` feature and can be installed and uninstalled using the normal processes described in Configuration.

WARNING	When this feature is installed, the Standard Catalog Framework feature `catalog-core-standardframework` should be uninstalled, as both catalog frameworks should not be installed simultaneously.

4.9.2. Configuring

The Fanout Catalog Framework can be configured using the normal processes described in Configuring DDF.

The configurable properties for the Fanout Catalog Framework are accessed from the Catalog Fanout Framework configuration in the Admin Console.

Configurable Properties

Title	Property	Type	Description	Default Value	Required
Default Timeout (in milliseconds)	`defaultTimeout`	Integer	The maximum amount of time to wait for a response from the Sources.	60000	yes

Title	Property	Type	Description	Default Value	Required
Product Cache Directory	`productCacheDirectory`	String	Directory where retrieved products will be cached for faster, future retrieval. If a directory path is specified with directories that do not exist, Catalog Framework will attempt to create those directories. Out of the box (without configuration), the product cache directory is `<INSTALL_DIR>/data/product-cache`. If a relative path is provided, it will be relative to the `<INSTALL_DIR>`. It is recommended to enter an absolute directory path, such as `/opt/product-cache` in Linux or `C:\product-cache` in Windows.	(empty)	no
Enable Product Caching	`cacheEnabled`	Boolean	Check to enable caching of retrieved products to provide faster retrieval for subsequent requests for the same product.	false	no
Delay (in seconds) between product retrieval retry attempts	`delayBetweenRetryAttempts`	Integer	The time to wait (in seconds) between attempting to retry retrieving a product.	10	no
Max product retrieval retry attempts	`maxRetryAttempts`	Integer	The maximum number of attempts to retry retrieving a product.	3	no
Caching Monitor Period	`cachingMonitorPeriod`	Integer	How many seconds to wait and not receive product data before retrying to retrieve a product.	5	no
Always Cache Product	`cacheWhenCanceled`	Boolean	Check to enable caching of retrieved products, even if client cancels the download.	false	no

Managed Service PID	`DDF.catalog.impl.service.fanout.FanoutCatalogFramework`
Managed Service Factory PID	N/A

Using

The Fanout Catalog Framework is a core class of DDF when configured as a fanout proxy. It provides the methods for query and resource retrieval operations on the Sources, where all operations are enterprise-wide operations. By contrast, the Standard Catalog Framework supports create/update/delete operations of metacards, in addition to the query and resource retrieval operations.

Use the Fanout Catalog Framework if:

- exposing a single node for enterprise access and hiding the details of the enterprise, such as federate source's names, is desired.
- access to individual federated sources is not required.
- access to a catalog provider to create, update, and delete metacards is not required.

The Fanout Catalog Framework also maintains a list of `ResourceReaders` for resource retrieval operations. A resource reader is matched to the scheme (i.e., protocol, such as `file://`) in the URI of the resource specified in the request to be retrieved.

Site information about the fanout configuration can be retrieved using the Fanout Catalog Framework. Site information includes the source's name, version, availability, and the list of unique content types currently stored in the source (e.g., NITF). Details of the individual federated sources is not included, only the fanout catalog framework.

Implementation Details

Exported Services

Registered Interface	Service Property	Value
`DDF.catalog.federation.FederationStrategy`	shortname	sorted
`org.osgi.service.event.EventHandler`	event.topics	`DDF/catalog/event/CREATED`, `DDF/catalog/event/UPDATED`, `DDF/catalog/event/DELETED`
`DDF.catalog.CatalogFramework`		
`DDF.catalog.event.EventProcessor`		
`DDF.catalog.plugin.PostIngestPlugin`		

Imported Services

Registered Interface	Availability	Multiple
DDF.cache.CacheManager		false
DDF.catalog.plugin.PostFederatedQueryPlugin	optional	true
DDF.catalog.plugin.PostIngestPlugin	optional	true
DDF.catalog.plugin.PostQueryPlugin	optional	true
DDF.catalog.plugin.PostResourcePlugin	optional	true
DDF.catalog.plugin.PreDeliveryPlugin	optional	true
DDF.catalog.plugin.PreFederatedQueryPlugin	optional	true
DDF.catalog.plugin.PreIngestPlugin	optional	true
DDF.catalog.plugin.PreQueryPlugin	optional	true
DDF.catalog.plugin.PreResourcePlugin	optional	true
DDF.catalog.plugin.PreSubscriptionPlugin	optional	true
DDF.catalog.plugin.PolicyPlugin	optional	true
DDF.catalog.plugin.AccessPlugin	optional	true
DDF.catalog.resource.ResourceReader	optional	true
DDF.catalog.source.ConnectedSource	optional	true
DDF.catalog.source.FederatedSource	optional	true
org.osgi.service.event.EventAdmin		false

4.9.3. Known Issues

None

4.9.4. Catalog Framework Camel Component

The catalog framework camel component supports creating, updating, and deleting metacards using the Catalog Framework from a Camel route.

URI Format

```
catalog:framework
```

Message Headers

Catalog Framework Producer

Header	Description
operation	the operation to perform using the catalog framework (possible values are CREATE | UPDATE | DELETE)

Sending Messages to Catalog Framework Endpoint

Catalog Framework Producer

In Producer mode, the component provides the ability to provide different inputs and have the Catalog framework perform different operations based upon the header values.

For the CREATE and UPDATE operation, the message body can contain a list of metacards or a single metacard object.

For the DELETE operation, the message body can contain a list of strings or a single string object. The string objects represent the IDs of metacards to be deleted. The exchange's "in" message will be set with the affected metacards. In the case of a CREATE, it will be updated with the created metacards. In the case of the UPDATE, it will be updated with the updated metacards and with the DELETE it will contain the deleted metacards.

Header	Message Body (Input)	Exchange Modification (Output)
operation = CREATE	List<Metacard> or Metacard	exchange.getIn().getBody() updated with List of Metacards created
operation = UPDATE	List<Metacard> or Metacard	exchange.getIn().getBody() updated with List of Metacards updated
operation = DELETE	List<String> or String (representing metacard IDs)	exchange.getIn().getBody() updated with List of Metacards deleted

Samples

This example demonstrates:

1. Reading in some sample data from the file system.
2. Using a Java bean to convert the data into a metacard.

3. Setting a header value on the Exchange.

4. Sending the Metacard to the Catalog Framework component for ingestion.

```
<route>
 <from uri="file:data/sampleData?noop=true  />
    <bean ref="sampleDataToMetacardConverter" method="covertToMetacard"/>\
   <setHeader headerName="operation">
  <constant>CREATE</constant>
 </setHeader>
    <to uri="catalog:framework"/>
</route>
```

4.9.5. Working with the Catalog Framework

Catalog Framework Reference

The Catalog Framework can be requested from the OSGi registry.

Blueprint Service Reference

```
<reference id="catalogFramework" interface="DDF.catalog.CatalogFramework" />
```

Methods

Create, Update, and Delete

Create, Update, and Delete (CUD) methods add, change, or remove stored metadata in the local Catalog Provider.

Create, Update, Delete Methods

```
public CreateResponse create(CreateRequest createRequest) throws IngestException,
SourceUnavailableException;
public UpdateResponse update(UpdateRequest updateRequest) throws IngestException,
SourceUnavailableException;
public DeleteResponse delete(DeleteRequest deleteRequest) throws IngestException,
SourceUnavailableException;
```

CUD operations process `PolicyPlugin`, `AccessPlugin`, and `PreIngestPlugin` instances before execution and `PostIngestPlugin` instances after execution.

Query

Query methods search metadata from available Sources based on the `QueryRequest` properties and Federation Strategy. Sources could include Catalog Provider, Connected Sources, and Federated

Sources.

Query Methods

```
public QueryResponse query(QueryRequest query) throws UnsupportedQueryException
,SourceUnavailableException, FederationException;
public QueryResponse query(QueryRequest queryRequest, FederationStrategy strategy) throws
SourceUnavailableException, UnsupportedQueryException, FederationException;
```

Query requests process `PolicyPlugin`, `AccessPlugin`, and `PreQueryPlugin` instances before execution and `PolicyPlugin`, `AccessPlugin`, and `PostQueryPlugin` instances after execution.

Resources

Resource methods retrieve products from Sources.

Resource Methods

```
public ResourceResponse getEnterpriseResource(ResourceRequest request) throwsIOException,
ResourceNotFoundException, ResourceNotSupportedException;
public ResourceResponse getLocalResource(ResourceRequest request) throws IOException,
ResourceNotFoundException, ResourceNotSupportedException;
public ResourceResponse getResource(ResourceRequest request, String resourceSiteName)
throws IOException, ResourceNotFoundException, ResourceNotSupportedException;
```

Resource requests process `PreResourcePlugin`s before execution and `PostResourcePlugin`s after execution.

Sources

Source methods can get a list of Source identifiers or request descriptions about Sources.

Source Methods

```
public Set<String> getSourceIds();
public SourceInfoResponse getSourceInfo(SourceInfoRequest sourceInfoRequest) throws
SourceUnavailableException;
```

Transforms

Transform methods provide convenience methods for using Metacard Transformers and Query Response Transformers.

Transform Methods

```
// Metacard Transformer
public BinaryContent transform(Metacard metacard, String transformerId, Map<String
,Serializable> requestProperties) throws CatalogTransformerException;

// Query Response Transformer
public BinaryContent transform(SourceResponse response, String transformerId, Map<String,
Serializable> requestProperties) throws CatalogTransformerException;
```

4.9.6. Developing Complementary Frameworks

DDF and the underlying OSGi technology can serve as a robust infrastructure for developing frameworks that complement the DDF Catalog.

Recommendations for Framework Development

1. Provide extensibility similar to that of the DDF Catalog.
    a. Provide a stable API with interfaces and simple implementations (refer to http://www.ibm.com/developerworks/websphere/techjournal/1007_charters/1007_charters.html).
2. Make use of the DDF Catalog wherever possible to store, search, and transform information.
3. Utilize OSGi standards wherever possible.
    a. ConfigurationAdmin
    b. MetaType
4. Utilize the sub-frameworks available in DDF.
    a. Karaf
    b. CXF
    c. PAX Web and Jetty

4.9.7. Developing Console Commands

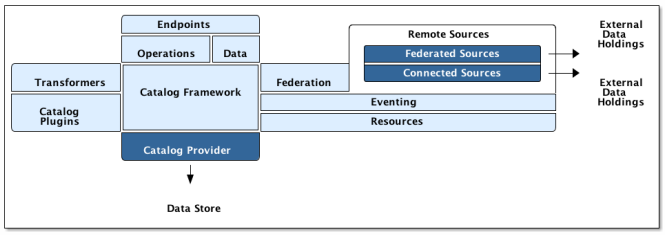
Console Commands

DDF supports development of custom console commands.

DDF includes custom commands for working with the Catalog, as described in the Managing Console Commands section.

4.10. Extending Sources

Catalog sources are used to connect Catalog components to data sources, local and remote. Sources act as proxies to the actual external data sources, e.g., a RDBMS database or a NoSQL database.



4.10.1. Existing Source Types

Catalog Provider

A Catalog provider provides an implementation of a searchable and writable catalog. All sources, including federated source and connected source, support queries, but a Catalog provider also allows metacards to be created, updated, and deleted.

A Catalog provider typically connects to an external application or a storage system (e.g., a database), acting as a proxy for all catalog operations.

Using

The Standard Catalog Framework uses only one Catalog provider, determined by the OSGi Framework as the service reference with the highest service ranking. In the case of a tie, the service with the lowest service ID (first created) is used.

The Catalog Fanout Framework App does not use a Catalog provider and will fail any create/update/delete operations even if there are active Catalog providers configured.

The Catalog reference implementation comes with a Solr Catalog Provider out of the box.

4.10.2. Remote Sources

Remote sources are read-only data sources that support query operations but cannot be used to create, update, or delete metacards.

TIP

Remote sources currently extend the `ResourceReader` interface. However, a `RemoteSource` is not treated as a `ResourceReader`. The `getSupportedSchemes()` method should never be called on a `RemoteSource`, thus the suggested implementation for a `RemoteSource` is to return an empty set. The `retrieveResource( )` and `getOptions( )` methods will be called and MUST be properly implemented by a `RemoteSource`.

4.10.3. Connected Source

A connected source is a remote source that is included in all local and federated queries but remains hidden from external clients. A connected source's identifier is removed in all query results by replacing it with DDF's source identifier. The Catalog Framework does not reveal a connected source as a separate source when returning source information responses.

image::query-flow.png, 500[]

4.10.4. Federated Source

A federated source is a remote source that can be included in federated queries by request or as part of an enterprise query. Federated sources support query and site information operations only. Catalog modification operations, such as create, update, and delete, are not allowed. Federated sources also expose an event service, which allows the Catalog Framework to subscribe to even notifications when metacards are created, updated, and deleted.

DDF Catalog instances can also be federated to each other. Therefore, a DDF Catalog can also act as a federated source to another DDF Catalog.

4.10.5. OpenSearch Source

The OpenSearch source provides a Federated Source that has the capability to do OpenSearch queries for metadata from Content Discovery and Retrieval (CDR) Search V1.1 compliant sources. The OpenSearch source does not provide a Connected Source interface.

Installing and Uninstalling

The OpenSearch source can be installed and uninstalled using the normal processes described in the Configuring DDF section.

Configuring

This component can be configured using the normal processes described in the Configuring DDF section. The configurable properties for the OpenSearch source are accessed from the Catalog OpenSearch Federated Source Configuration in the Web Console.

Configuring the OpenSearch Source

Configurable Properties

Title	Property	Type	Description	Default Value	Required
Source Name	`shortname`	String		DDF-OS	Yes

Title	Property	Type	Description	Default Value	Required
OpenSearch service URL	`endpointUrl`	String	The OpenSearch endpoint URL, e.g., DDF's OpenSearch endpoint (http://0.0.0.0:8181/services/catalog/query?q={searchTerms}...)	https://example.com?q={searchTerms}&src={fs:routeTo?}&mr={fs:maxResults?}&count={count?}&mt={fs:maxTimeout?}&dn={idn:userDN?}&lat={geo:lat?}&lon={geo:lon?}&radius={geo:radius?}&bbox={geo:box?}&polygon={geo:polygon?}&dtstart={time:start?}&dtend={time:end?}&dateName={cat:dateName?}&filter={fsa:filter?}&sort={fsa:sort?}	Yes
Username	`username`	String	Username to use with HTTP Basic Authentication. This auth info will overwrite any federated auth info. Only set this if the OpenSearch endpoint requires basic authentication.		No
Password	`password`	String	Password to use with HTTP Basic Authentication. This auth info will overwrite any federated auth info. Only set this if the OpenSearch endpoint requires basic authentication.		No

Title	Property	Type	Description	Default Value	Required
Always perform local query	`localQueryOnly`	Boolean	Always performs a local query by setting src=local OpenSearch parameter in endpoint URL. **This must be set if federating to another DDF**.	false	Yes
Convert to BBox	`shouldConvertToBBox`	Boolean	Converts Polygon and Point-Radius searches to a Bounding Box for compatibility with legacy interfaces. Generated bounding box is a very rough representation of the input geometry	true	Yes

Using

Use the OpenSearch source if querying a CDR-compliant search service is desired.

Query Format

OpenSearch Parameter to DDF Query Mapping

OpenSearch/CDR Parameter	DDF Data Location
q={searchTerms}	Pulled verbatim from DDF query.
src={fs:routeTo?}	Unused
mr={fs:maxResults?}	Pulled verbatim from DDF query.
count={count?}	Pulled verbatim from DDF query.
mt={fs:maxTimeout?}	Pulled verbatim from DDF query.
dn={idn:userDN?}	DDF Subject
lat={geo:lat?}	Pulled verbatim from DDF query.
lon={geo:lon?}	Pulled verbatim from DDF query.
radius={geo:radius?}	Pulled verbatim from DDF query.
bbox={geo:box?}	Converted from Point-Radius DDF query.
polygon={geo:polygon?}	Pulled verbatim from DDF query.
dtstart={time:start?}	Pulled verbatim from DDF query.
dtend={time:end?}	Pulled verbatim from DDF query.
dateName={cat:dateName?}	Unused
filter={fsa:filter?}	Unused

OpenSearch/CDR Parameter	DDF Data Location
sort={fsa:sort?}	Translated from DDF query. Format: "relevance" or "date" Supports "asc" and "desc" using colon as delimiter.

Implementation Details

Exported Services

Registered Interface	Service Property	Value
`DDF.catalog.source.FederatedSource`		

Imported Services

Registered Interface	Availability	Multiple	Filter
DDF.catalog.transform.InputTransformer	required	false	(&(mime-type=text/xml)(id=xml))

Known Issues

The OpenSearch source does not provide a Connected Source interface.

4.10.6. Developing a Source

Sources are components that enable DDF to talk to back-end services. They let DDF perform query and ingest operations on catalog stores and query operations on federated sources. Sources reside in the Sources area of the DDF Overview.

Creating a New Source

Implement a Source Interface

There are three types of sources that can be created. All of these types of sources can perform a query operation. Operating on queries is the foundation for all sources. All of these sources must also be able to return their availability and the list of content types currently stored in their back-end data stores.

- Catalog Provider - `DDF.catalog.source.CatalogProvider`
  Used to communicate with back-end storage. Allows for Query and Create/Update/Delete operations.
- Federated Source - `DDF.catalog.source.FederatedSource`
  Used to communicate with remote systems. Only allows query operations.
- Connected Source - `DDF.catalog.source.ConnectedSource`
  Similar to a Federated Source with the following exceptions:

Queried on all local queries

- *SiteName is hidden (masked with the DDF sourceId) in query results*
- *SiteService does not show this Source's information separate from DDF's.*

The procedure for implementing any of the source types follows a similar format: . Create a new class that implements the specified Source interface and `ConfiguredService`. . Implement the required methods. . Create an OSGi descriptor file to communicate with the OSGi registry. (Refer to the OSGi Services section.) .. Import DDF packages. .. Register source class as service to the OSGi registry. . Deploy to DDF.

IMPORTANT	The `factory-pid` property of the metatype must contain one of the following in the name: service, Service, source, Source

Catalog Provider

1. Create a Java class that implements `CatalogProvider`.
   `public class TestCatalogProvider implements DDF.catalog.source.CatalogProvider`
2. Implement the required methods from the `DDF.catalog.source.CatalogProvider` interface.
   `public CreateResponse create(CreateRequest createRequest) throws IngestException; public UpdateResponset update(UpdateRequest updateRequest) throws IngestException; public DeleteResponse delete(DeleteRequest deleteRequest) throws IngestException;`
3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: DDF.catalog, DDF.catalog.source`
4. Export the service to the OSGi registry.

Blueprint example

```
<service ref="[[TestCatalogProvider]]" interface="DDF.catalog.source.CatalogProvider" />
```

The DDF Integrator's Guide provides details on the following Catalog Providers that come with DDF out of the box.

NOTE	A code example of a Catalog Provider delivered with DDF is the Catalog Solr Embedded Provider.

Federated Source

1. Create a Java class that implements `FederatedSource` and `ConfiguredService`.
   `public class TestFederatedSource implements DDF.catalog.source.FederatedSource, DDF.catalog.service.ConfiguredService`
2. Implement the required methods of the `DDF.catalog.source.FederatedSource` and

`DDF.catalog.service.ConfiguredService` interfaces.

3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: DDF.catalog, DDF.catalog.source`

4. Export the service to the OSGi registry.

Blueprint example

```
<service ref="[[TestFederatedSource]]" interface="DDF.catalog.source.FederatedSource" />
```

NOTE	A code example of a Federated Source delivered with DDF can be found in `DDF.catalog.source.solr`

Connected Source

1. Create a Java class that implements `ConnectedSource` and `ConfiguredService`.
   `public class TestConnectedSource implements DDF.catalog.source.ConnectedSource, DDF.catalog.service.ConfiguredService`

2. Implement the required methods of the `DDF.catalog.source.ConnectedSource` and `DDF.catalog.service.ConfiguredService` interfaces.

3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: DDF.catalog, DDF.catalog.source`

4. Export the service to the OSGi registry.

Blueprint example

```
<service ref="[[TestConnectedSource]]" interface="DDF.catalog.source.ConnectedSource" />
```

IMPORTANT	In some Providers that are created, there is a need to make Web Service calls through JAXB clients. It is best NOT to create your JAXB client as a global variable. There may be intermittent failures with the creation of Providers and federated sources when clients are created in this manner. Create your JAXB clients every single time within the methods that require it in order to avoid this issue.

Exception Handling

In general, sources should only send information back related to the call, not implementation details.

Examples

- "Site XYZ not found" message rather than the full stack trace with the original site not found exception.
- The caller issues a malformed search request. Return an error describing the right form, or specifically what was not recognized in the request. Do not return the exception and stack trace where the parsing broke.
- The caller leaves something out. Do not return the null pointer exception with a stack trace, rather return a generic exception with the message "xyz was missing."

Additional Information

- Three Rules for Effective Exception Handling

4.10.7. Developing a Filter Delegate

Filter Delegates help reduce the complexity of parsing OGC Filters. The reference Filter Adapter implementation contains the necessary boilerplate visitor code and input normalization to handle commonly supported OGC Filters.

Creating a New Filter Delegate

A Filter Delegate contains the logic that converts normalized filter input into a form that the targeted data source can handle. Delegate methods will be called in a depth first order as the Filter Adapter visits filter nodes.

Implementing the Filter Delegate

1. Create a Java class extending `FilterDelegate`.
   ```
   public class ExampleDelegate extends DDF.catalog.filter.FilterDelegate<ExampleReturnObjectType>
   {
   ```
2. `FilterDelegate` will throw an appropriate exception for all methods not implemented. Refer to the DDF JavaDoc for more details about what is expected of each `FilterDelegate` method.

NOTE	A code example of a Filter Delegate can be found in `DDF.catalog.filter.proxy.adapter.test` of the `filter-proxy` bundle.

Throwing Exceptions

Filter delegate methods can throw `UnsupportedOperationException` run-time exceptions. The `GeotoolsFilterAdapterImpl` will catch and re-throw these exceptions as `UnsupportedQueryExceptions`.

Using the Filter Adapter

The FilterAdapter can be requested from the OSGi registry.

```
<reference id="filterAdapter" interface="DDF.catalog.filter.FilterAdapter" />
```

The Query in a QueryRequest implements the Filter interface. The Query can be passed to a `FilterAdapter` and `FilterDelegate` to process the Filter.

```
@Override
public DDF.catalog.operation.QueryResponse query(DDF.catalog.operation.QueryRequest
queryRequest)
    throws DDF.catalog.source.UnsupportedQueryException {

    DDF.catalog.operation.Query query = queryRequest.getQuery();

    DDF.catalog.filter.FilterDelegate<ExampleReturnObjectType> delegate = new
ExampleDelegate();

    // DDF.catalog.filter.FilterAdapter adapter injected via Blueprint
    ExampleReturnObjectType result = adapter.adapt(query, delegate);
}
```

Import the DDF Catalog API Filter package and the reference implementation package of the Filter Adapter in the bundle manifest (in addition to any other required packages).
`Import-Package: DDF.catalog, DDF.catalog.filter, DDF.catalog.source`

Filter Support

Not all OGC Filters are exposed at this time. If demand for further OGC Filter functionality is requested, it can be added to the Filter Adapter and Delegate so sources can support more complex filters. The following OGC Filter types are currently available:

Logical
And
Or
Not
Include
Exclude

Property Comparison
`PropertyIsBetween`
`PropertyIsEqualTo`
`PropertyIsGreaterThan`

Property Comparison
PropertyIsGreaterThanOrEqualTo
PropertyIsLessThan
PropertyIsLessThanOrEqualTo
PropertyIsLike
PropertyIsNotEqualTo
PropertyIsNull

Spatial	Definition
Beyond	True if the geometry being tested is beyond the stated distance of the geometry provided.
Contains	True if the second geometry is wholly inside the first geometry.
Crosses	True if the intersection of the two geometries results in a value whose dimension is less than the geometries and the maximum dimension of the intersection value includes points interior to both the geometries, and the intersection value is not equal to either of the geometries.
Disjoint	True if the two geometries do not touch or intersect.
DWithin	True if the geometry being tested is within the stated distance of the geometry provided.
Intersects	True if the two geometries intersect. This is a convenience method as you could always ask for Not Disjoint(A,B) to get the same result.
Overlaps	True if the intersection of the geometries results in a value of the same dimension as the geometries that is different from both of the geometries.
Touches	True if and only if the only common points of the two geometries are in the union of the boundaries of the geometries.
Within	True if the first geometry is wholly inside the second geometry.

Temporal
After
Before
During

4.11. Extending Catalog Transformers

Transformers transform data to and from various formats. Transformers can be categorized on the basis of when they are invoked and used. The existing types are Input transformers, Metacard

transformers, and Query Response transformers. Additionally, XSLT transformers are provided to aid in developing custom, lightweight Metacard and Query Response transformers.

Transformers are utility objects used to transform a set of standard DDF components into a desired format, such as into PDF, GeoJSON, XML, or any other format. For instance, a transformer can be used to convert a set of query results into an easy-to-read GeoJSON format (GeoJSON Transformer) or convert a set of results into a RSS feed that can be easily published to a URL for RSS feed subscription. A major benefit of transformers is that they can be registered in the OSGi Service Registry so that any other developer can access them based on their standard interface and self-assigned identifier, referred to as its "shortname." Transformers are often used by endpoints for data conversion in a system standard way. Multiple endpoints can use the same transformer, a different transformer, or their own published transformer.

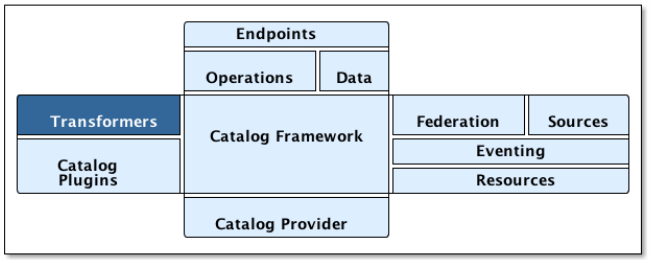


Figure 8. Transformers

WARNING	The current transformers only work for UTF-8 characters and do not support Non-Western Characters (e.g., Hebrew). It is recommend not to use international character sets as they may not be displayed properly.

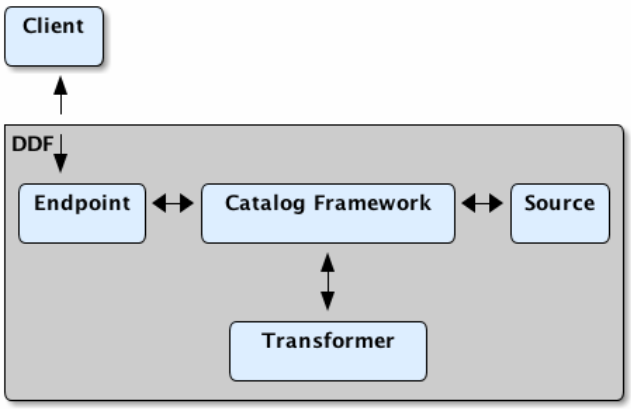


Figure 9. Communication Diagram

4.11.1. Working with Transformers

The `DDF.catalog.transform` package includes the `InputTransformer`, `MetacardTransformer`, and `QueryResponseTransformer` interfaces. All implementations can be accessed using the Catalog Framework or OSGi Service Registry, as long as the implementations have been registered with the

Service Registry.

4.11.2. Catalog Framework

The `CatalogFramework` provides convenient methods to transform `Metacards` and `QueryResponses` using a reference to the `CatalogFramework`. See Working with the Catalog Framework for more details on the method signatures. It is easy to execute the convenience `transform` methods on the `CatalogFramework` instance.

Query Response Transform Example

```
// inject CatalogFramework instance or retrieve an instance
private CatalogFramework catalogFramework;

public RSSEndpoint(CatalogFramework catalogFramework)
{
     this.catalogFramework = catalogFramework ;
     // implementation
}

 // Other implementation details ...

private void convert(QueryResponse queryResponse ) {
    // ...
    String transformerId = "rss";

    BinaryContent content = catalogFramework.transform(queryResponse, transformerId,
null);

    // ...

}
```

Line #	Action
4	`CatalogFramework` is injected, possibly by dependency injection framework.
16	`queryResponse` is transformed into the RSS format, which is stored in the `BinaryContent` instance

4.11.3. Dependency Injection

Using Blueprint or another injection framework, transformers can be injected from the OSGi Service Registry.

Blueprint Service Reference

```
<reference id="[[Reference Id]]" interface="DDF.catalog.transform.[[Transformer Interface
Name]]" filter="(shortname=[[Transformer Identifier]])" />
```

Each transformer has one or more `transform` methods that can be used to get the desired output.

Input Transformer Example

```
DDF.catalog.transform.InputTransformer inputTransformer = retrieveInjectedInstance() ;

Metacard entry = inputTransformer.transform(messageInputStream);
```

Metacard Transformer Example

```
DDF.catalog.transform.MetacardTransformer metacardTransformer = retrieveInjectedInstance
() ;

BinaryContent content = metacardTransformer.transform(metacard, arguments);
```

Query Response Transformer Example

```
DDF.catalog.transform.QueryResponseTransformer queryResponseTransformer =
retrieveInjectedInstance() ;

BinaryContent content = queryResponseTransformer.transform(sourceSesponse, arguments);
```

4.11.4. OSGi Service Registry

IMPORTANT	In the vast majority of cases, working with the OSGi Service Reference directly should be avoided. Instead, dependencies should be injected via a dependency injection framework like Blueprint.

Transformers are registered with the OSGi Service Registry. Using a `BundleContext` and a filter, references to a registered service can be retrieved.

OSGi Service Registry Reference Example

```
ServiceReference[] refs =
    bundleContext.getServiceReferences(DDF.catalog.transform.InputTransformer.class.getNa
me(),"(shortname=" + transformerId + ")");
InputTransformer inputTransformer = (InputTransformer) context.getService(refs[0]);
Metacard entry = inputTransformer.transform(messageInputStream);
```

4.11.5. Included Input Transformers

An input transformer transforms raw data (text/binary) into a Metacard.

Once converted to a Metacard, the data can be used in a variety of ways, such as in an `UpdateRequest`, `CreateResponse`, or within Catalog Endpoints or Sources. For instance, an input transformer could be used to receive and translate XML into a Metacard so that it can be placed within a `CreateRequest` in order to be ingested within the Catalog. Input transformers should be registered within the Service Registry with the interface `DDF.catalog.transform.InputTransformer` in order to notify some Catalog components of any new transformers.

Tika Input Transformer

The Tika Input Transformer is the default input transformer responsible for translating Microsoft Word, Microsoft Excel, Microsoft PowerPoint, OpenOffice Writer, and PDF documents into a Catalog Metacard. This input transformer utilizes Apache Tika to provide basic support for these mime types. As such, the metadata extracted from these types of documents is the metadata that is common across all of these document types, e.g., creation date, author, last modified date, etc. The Tika Input Transformer's main purpose is to ingest these types of content into the Metadata Catalog.

The Tika input transformer is given a service ranking (prioity) of -1 so that it is guaranteed to be the last input transformer that is invoked. This allows any registered input transformer that are more specific for any of these document types to be invoked instead of this rudimentary default input transformer.

Installing and Uninstalling

This transformer is installed by default. To install or uninstall manually, use the `tika-input-transformer` feature in the Admin Console (https://localhost:8993/admin) under DDF Catalog Features or use the System Console.

Install the `catalog-transformer-tika` feature using the Admin Console. This feature is uninstalled by default.

Configuring

None

Using

Use the Tika Input Transformer for ingesting Microsoft documents, OpenOffice documents, or PDF documents into the Catalog.

Service Properties

Key	Value
mime-type	* application/pdf * application/vnd.openxmlformats-officedocument.wordprocessingml.document * application/vnd.openxmlformats-officedocument.spreadsheetml.sheet * application/vnd.openxmlformats-officedocument.presentationml.presentation * application/vnd.openxmlformats-officedocument.presentationml.presentation * application/vnd.ms-powerpoint.presentation.macroenabled.12 * application/vnd.ms-powerpoint.slideshow.macroenabled.12 * application/vnd.openxmlformats-officedocument.presentationml.slideshow * application/vnd.ms-powerpoint.template.macroenabled.12 * application/vnd.oasis.opendocument.text
shortname	
id	tika
title	Tika Input Transformer
description	Default Input Transformer for all mime types.
`service.ranking`	-1

Implementation Details

This input transformer maps the metadata common across all mime types to applicable metacard attributes in the default MetacardType.

PPTX Input Transformer

The PPTX Input Transformer is the input transformer responsible for translating Microsoft PowerPoint (OOXML only) documents into a Catalog Metacard. This input transformer utilizes Apache Tika (for basic metadata) and Apache POI (for thumbnail creation). The PPTX Input Transformer's main purpose is to ingest PPTX documents into the DDF Content Repository and the Metadata Catalog.

The PPTX Input Transformer will take precedence over the Tika Input Transformer for PPTX documents.

Installing and Uninstalling

This transformer is installed by default. To install or uninstall manually, use the `catalog-transformer-pptx` feature in the Admin Console (https://localhost:8993/admin) under DDF Catalog Features.

Configuring

This transformer does not require any configuring.

Using

Use the PPTX Input Transformer for ingesting Microsoft PowerPoint (OOXML only) documents into the DDF Content Repository and/or the Metadata Catalog.

Service Properties

Key	Value
mime-type	* application/vnd.openxmlformats-officedocument.presentationml.presentation
id	pptx
title	PPTX Input Transformer
description	Default Input Transformer for the `application/vnd.openxmlformats-officedocument.presentationml.presentation` mime type.

Implementation Details

This input transformer maps the metadata common across all mime types to applicable metacard attributes in the default MetacardType and adds a thumbnail of the first page in the PPTX document.

Video Input Transformer

The video input transformer is responsible for creating Catalog metacards from certain video file types. Currently, it is responsible for handling MPEG-2 transport streams as well as MPEG-4, AVI, MOV, and WMV videos. This input transformer uses Apache Tika to extract basic metadata from the video files and applies more sophisticated methods to extract more meaningful metadata from these types of video.

Installing and Uninstalling

This transformer is installed by default. To install or uninstall manually, use the `video-input-transformer` feature in the Admin Console (https://localhost:8993/admin) under DDF Catalog Features or use the System Console.

Configuring

None

Using

Use the video input transformer for ingesting video files into the Catalog.

Service Properties

Key	Value
mime-type	* video/avi * video/msvideo * video/vnd.avi * video/x-msvideo * video/mp4 * video/MP2T * video/mpeg * video/quicktime * video/wmv * video/x-ms-wmv
shortname	video
id	video
description	Detects and extracts metadata from various video file formats.

GeoJSON Input Transformer

The GeoJSON input transformer is responsible for translating specific GeoJSON into a Catalog metacard.

Installing and Uninstalling

Install the `catalog-rest-endpoint` feature using the https://localhost:8993/admin.

Configuring

None

Using

Using the REST Endpoint, for example, HTTP POST a GeoJSON metacard to the Catalog. Once the REST Endpoint receives the GeoJSON Metacard, it is converted to a Catalog metacard.

Example HTTP POST of a local `metacard.json` file using the Curl Command

```
curl -X POST -i -H "Content-Type: application/json" -d "@metacard.json"
https://localhost:8993/services/catalog
```

Conversion

A GeoJSON object consists of a single JSON object. The single JSON object can be a geometry, a feature, or a FeatureCollection. This input transformer only converts "feature" objects into metacards. This is a natural choice since feature objects include geometry information and a list of properties. For instance, if only a geometry object is passed, such as only a LineString, that is not enough information to create a metacard. This input transformer currently does not handle FeatureCollections either, but could be

supported in the future.

IMPORTANT

Cannot create Metacard from this limited GeoJSON

```
{ "type": "LineString",
 "coordinates": [ [100.0, 0.0], [101.0, 1.0] ]
 }
```

The following sample *will* create a valid metacard:

Sample Parseable GeoJson (Point)

```
{
    "properties": {
        "title": "myTitle",
        "thumbnail": "CA==",
        "resource-uri": "http://example.com",
        "created": "2012-09-01T00:09:19.368+0000",
        "metadata-content-type-version": "myVersion",
        "metadata-content-type": "myType",
        "metadata": "<xml></xml>",
        "modified": "2012-09-01T00:09:19.368+0000"
    },
    "type": "Feature",
    "geometry": {
        "type": "Point",
        "coordinates": [
            30.0,
            10.0
        ]
    }
}
```

In the current implementation, `Metacard.LOCATION` is not taken from the properties list as WKT, but instead interpreted from the `geometry` JSON object. The geometry object is formatted according to the GeoJSON standard. Dates are in the ISO 8601 standard. White space is ignored, as in most cases with JSON. Binary data is accepted as Base64. XML must be properly escaped, such as what is proper for normal JSON.

Only Required Attributes are recognized in the properties currently.

Metacard Extensibility

GeoJSON supports custom, extensible properties on the incoming GeoJSON using DDF's extensible metacard support. To have those customized attributes understood by the system, a corresponding `MetacardType` must be registered with the `MetacardTypeRegistry`. That `MetacardType` must be specified by name in the metacard-type property of the incoming GeoJSON. If a `MetacardType` is specified on the

GeoJSON input, the customized properties can be processed, cataloged, and indexed.

```
{
    "properties": {
        "title": "myTitle",
        "thumbnail": "CA==",
        "resource-uri": "http://example.com",
        "created": "2012-09-01T00:09:19.368+0000",
        "metadata-content-type-version": "myVersion",
        "metadata-content-type": "myType",
        "metadata": "<xml></xml>",
        "modified": "2012-09-01T00:09:19.368+0000",
        "min-frequency": "10000000",
        "max-frequency": "20000000",
        "metacard-type": "DDF.metacard.custom.type"
 },
    "type": "Feature",
    "geometry": {
        "type": "Point",
        "coordinates": [
            30.0,
            10.0
        ]
    }
}
```

When the GeoJSON Input Transformer gets GeoJSON with the `MetacardType` specified, it will perform a lookup in the `MetacardTypeRegistry` to obtain the specified `MetacardType` in order to understand how to parse the GeoJSON. If no `MetacardType` is specified, the GeoJSON Input Transformer will assume the default `MetacardType`. If an unregistered `MetacardType` is specified, an exception will be returned to the client indicating that the `MetacardType` was not found.

Package Details

Feature Information

N/A

Included Bundles

N/A

Services

Exported Services

Table 7. DDF.catalog.transform.InputTransformer

mime-type	application/json
id	geojson

Implementation Details

Table 8. Exported Services

Registered Interface	Service Property	Value
`DDF.catalog.transform.InputTransformer`	mime-type	application/json
	id	geojson

Known Issues

Does not handle multiple geometries yet.

4.11.6. Developing an Input Transformer

Using Java

1. Create a new Java class that implements DDF.catalog.transform.InputTransformer.
   ```
   public class SampleInputTransformer implements DDF.catalog.transform.InputTransformer
   ```
2. Implement the transform methods.
   ```
   public Metacard transform(InputStream input) throws IOException, CatalogTransformerException
   public Metacard transform(InputStream input, String id) throws IOException, CatalogTransformerException
   ```
3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   ```
   Import-Package: DDF.catalog,DDF.catalog.transform
   ```
4. Create an OSGi descriptor file to communicate with the OSGi Service Registry (described in the Working with OSGi section). Export the service to the OSGi Registry and declare service properties.

Blueprint descriptor example

```
...
<service ref="[[SampleInputTransformer]]" interface=
"DDF.catalog.transform.InputTransformer">
    <service-properties>
        <entry key="shortname" value="[[sampletransform]]" />
        <entry key="title" value="[[Sample Input Transformer]]" />
        <entry key="description" value="[[A new transformer for metacard input.]]" />
    </service-properties>
</service>
...
```

1. Deploy OSGi Bundle to OSGi runtime.

Variable Descriptions

Table 9. Blueprint Service Properties

Key	Description of Value	Example
`shortname`	(Required) An abbreviation for the return-type of the BinaryContent being sent to the user.	*atom*
`title`	(Optional) A user-readable title that describes (in greater detail than the shortname) the service.	*Atom Entry Transformer Service*
`description`	(Optional) A short, human-readable description that describes the functionality of the service and the output.	*This service converts a single metacard xml document to an atom entry element.*

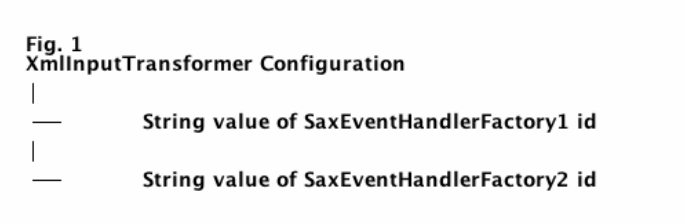
Create an XML Input Transformer using SaxEventHandlers

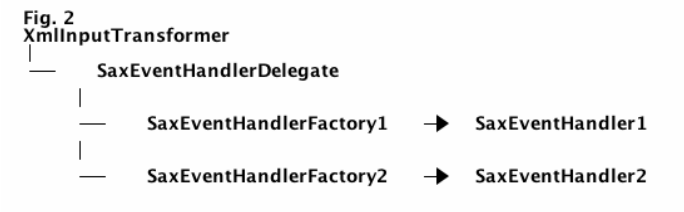
If the transformer will transform XML, (as opposed to JSON or a Word document, for example) there is a simpler solution than fully implementing a MetacardValidator. DDF includes an extensible, configurable `XmlInputTransformer`. This transformer can be instantiated via blueprint as a managed service factory and configured via metatype. The `XmlInputTransformer` takes a configuration of `SaxEventHandler`s. A `SaxEventHandler` is a class that handles SAX Events (a very fast XML parser) to parse metadata and create metacards. As many `SaxEventHandler`s as needed can be implemented and included in the `XmlInputTransformer` configuration. See the `catalog-transformer-streaming-impl` bundle for examples (`XmlSaxEventHandlerImpl` which parses the DDF Metacard XML Metadata and the `GmlHandler` which parses GML 2.0) Each `SaxEventHandler` implementation has a `SaxEventHandlerFactory` associated with it. The `SaxEventHandlerFactory` is responsible for instantiating new `SaxEventHandlers` - each transform request gets a new instance of `XmlInputTransformer` and set of `SaxEventHandler`s to be thread- and state-safe.

The following diagrams intend to clarify implementation details:

Figure 1 shows the XmlInputTransformer configuration, which is configured using the metatype and

has the SaxEventHandlerFactory ids. Then, when a transform request is received, the ManagedServiceFactory instantiates a new XmlInputTransformer. This XmlInputTransformer then instantiates a new SaxEventHandlerDelegate with the configured SaxEventHandlersFactory ids. The factories all in turn instantiate a SaxEventHandler. Then, the SaxEventHandlerDelegate begins parsing the XML input document, handing the SAX Events off to each SaxEventHandler, which handle them if they can. After parsing is finished, each SaxEventHandler returns a list of Attributes to the SaxEventHandlerDelegate and XmlInputTransformer which add the attributes to the metacard and then return the fully constructed metacard.





For more specific details, see the Javadoc for the org.codice.ddf.transformer.xml.streaming.* package. Additionally, see the source code for the org.codice.ddf.transformer.xml.streaming.impl.GmlHandler.java, org.codice.ddf.transformer.xml.streaming.impl.GmlHandlerFactory, org.codice.ddf.transformer.xml.streaming.impl.XmlInputTransformerImpl, and org.codice.ddf.transformer.xml.streaming.impl.XmlInputTransformerImplFactory.

NOTE

1. The XmlInputTransformer & SaxEventHandlerDelegate create and configure themselves based on String matches of the configuration ids with the SaxEventHandlerFactory ids, so ensure these match.

2. The XmlInputTransformer uses a DynamicMetacardType. This is pertinent because a metacards attributes are only stored in Solr if they are declared on the MetacardType. Since the DynamicMetacardType is constructed dynamically, attributes are declared by the SaxEventHandlerFactory that parses them, as opposed to the MetacardType. See org.codice.ddf.transformer.xml.streaming.impl.XmlSaxEventHandlerFactoryImpl.java vs ddf.catalog.data.impl.BasicTypes.java

Create an Input Transformer Using Apache Camel

Alternatively, make an Apache Camel route in a blueprint file and deploy it using a feature file or via hot deploy.

Design Pattern

From

When using **from** `catalog:inputtransformer?id=text/xml`, an Input Transformer will be created and registered in the OSGi registry with an id of `text/xml`.

To

When using **to** `catalog:inputtransformer?id=text/xml`, an Input Transformer with an id matching text/xml will be discovered from the OSGi registry and invoked.

Message Formats

Table 10. InputTransformer

Exchange Type	Field	Type
Request (comes from `<from>` in the route)	body	java.io.InputStream
Response (returned after called via `<to>` in the route)	body	`DDF.catalog.data.Metacard`

Examples

InputTransformer Creation

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
        <route>
            <from uri="catalog:inputtransformer?mimeType=RAW(id=text/xml;id=vehicle)"/>
            <to uri="xslt:vehicle.xslt" /> <!-- must be on classpath for this bundle -->
            <to uri=
"catalog:inputtransformer?mimeType=RAW(id=application/json;id=geojson)" />
        </route>
    </camelContext>
</blueprint>
```

TIP Its always a good idea to wrap the `mimeType` value with the RAW parameter as shown in the example above. This will ensure that the value is taken exactly as is, and is especially useful when you are using special characters.

Line Number	Description
1	Defines this as an Apache Aries blueprint file.
2	Defines the Apache Camel context that contains the route.

Line Number	Description
3	Defines start of an Apache Camel route.
4	Defines the endpoint/consumer for the route. In this case it is the DDF custom catalog component that is an InputTransformer registered with an id of text/xml;id=vehicle meaning it can transform an InputStream of vehicle data into a metacard. **Note that the specified XSL stylesheet must be on the classpath of the bundle that this blueprint file is packaged in.**
5	Defines the XSLT to be used to transform the vehicle input into GeoJSON format using the Apache Camel provided XSLT component.

NOTE An example of using an Apache Camel route to define an `InputTransformer` in a blueprint file and deploying it as a bundle to an OSGi container can be found in the DDF SDK examples at `DDF/sdk/sample-transformers/xslt-identity-input-transformer`

4.11.7. Included Metacard InputTransformers

A metacard transformer transforms a metacard into other data formats.

HTML Metacard Transformer

The HTML metacard transformer is responsible for translating a metacard into an HTML formatted document.

Installing and Uninstalling

Install the `catalog-transformer-html` feature using the Admin Console.

Configuring

None

Using

Using the REST Endpoint for example, request a metacard with the transform option set to the HTML shortname.

```
http://localhost:8181/services/catalog/0123456789abcdef0123456789abcdef?transform=html
```

Example Output

```
html metacard.png
```

Implementation Details

Registered Interface	Service Property	Value
DDF.catalog.transform.MetacardTransformer	title	View as html...
	description	Transforms query results into html
	shortname (for backwards compatibility)	html

Known Issues

None

4.11.8. XML Metacard Transformer

The XML metacard transformer is responsible for translating a metacard into an XML-formatted document. The metacard element that is generated is an extension of `gml:AbstractFeatureType`, which makes the output of this transformer GML 3.1.1 compatible.

Installing and Uninstalling

This transformer comes installed out of the box and is running on startup. To install or uninstall manually, use the `catalog-transformer-xml` feature.

Configuring

None

Using

Using the REST Endpoint for example, request a metacard with the transform option set to the XML shortname.

```
https://localhost:8993/services/catalog/ac0c6917d5ee45bfb3c2bf8cd2ebaa67?transform=xml
```

Implementation Details

Metacard to XML Mappings

Metacard Variables
XML Element
id
metacard/@gml:id

Metacard Variables
`metacardType`
`metacard/type`
`sourceId`
`metacard/source`
`all other attributes`

`metacard/<AttributeType>[name='<AttributeName>']/value`
 	
`For instance, the value for the metacard attribute named "title" would be found at metacard/string[@name='title']/value`

AttributeTypes

XML Adapted Attributes
boolean
base64Binary
dateTime
double
float
geometry
int
long
object
short
string
stringxml

Known Issues

None

4.11.9. GeoJSON Metacard Transformer

GeoJSON Metacard Transformer translates a Metacard into GeoJSON.

Installing and Uninstalling

Install the `catalog-transformer-json` feature.

Configuring

None

Using

The GeoJSON Metacard Transformer can be used programmatically by requesting a `MetacardTransformer` with the id `geojson`. It can also be used within the REST Endpoint by providing the transform option as `geojson`.

Example REST GET method with the GeoJSON MetacardTransformer

```
http://localhost:8181/services/catalog/0123456789abcdef0123456789abcdef?transform=geojson
```

Example Output

```
{
    "properties":{
        "title":"myTitle",
        "thumbnail":"CA==",
        "resource-uri":"http:\/\/example.com",
        "created":"2012-08-31T23:55:19.518+0000",
        "metadata-content-type-version":"myVersion",
        "metadata-content-type":"myType",
        "metadata":"<xml>text<\/xml>",
        "modified":"2012-08-31T23:55:19.518+0000",
        "metacard-type": "DDF.metacard"
    },
    "type":"Feature",
    "geometry":{
        "type":"LineString",
        "coordinates":[
            [
                30.0,
                10.0
            ],
            [
                10.0,
                30.0
            ],
            [
                40.0,
                40.0
            ]
        ]
    }
}
```

Implementation Details

Registered Interface	Service Property	Value
DDF.catalog.transform.MetacardTransformer	mime-type	application/json
	id	geojson
	shortname (for backwards compatibility)	geojson

4.11.10. Known Issues

None

4.11.11. Thumbnail Metacard Transformer

The Thumbnail Metacard Transformer retrieves the thumbnail bytes of a Metacard by returning the `Metacard.THUMBNAIL` attribute value.

Installing and Uninstalling

This transformer is installed out of the box. To uninstall the transformer, you must stop or uninstall the bundle.

Configuring

None

Using

Endpoints or other components can retrieve an instance of the Thumbnail Metacard Transformer using its id `thumbnail`.

Sample Blueprint Reference Snippet

```
<reference id="metacardTransformer" interface="DDF.catalog.transform.MetacardTransformer"
filter="(id=thumbnail)"/>
```

The Thumbnail Metacard Transformer returns a `BinaryContent` object of the `Metacard.THUMBNAIL` bytes and a MIME Type of `image/jpeg`.

Implementation Details

Service Property	Value
id	thumbnail
shortname	thumbnail

mime-type	image/jpeg

Known Issues

None

4.11.12. Metadata Metacard Transformer

The Metadata Metacard Transformer returns the `Metacard.METADATA` attribute when given a metacard. The MIME Type returned is `text/xml`.

Installing and Uninstalling

Catalog Transformers application will install this feature when deployed. This transformer's feature, `catalog-transformer-metadata`, can be uninstalled or installed.

Configuring

None

Using

The Metadata Metacard Transformer can be used programmatically by requesting a MetacardTransformer with the id metadata. It can also be used within the REST Endpoint by providing the transform option as metadata.

Example REST GET method with the Metadata MetacardTransformer

```
http://localhost:8181/services/catalog/0123456789abcdef0123456789abcdef?transform=metadata
```

Implementation Details

Registered Interface	Service Property	Value
`DDF.catalog.transform.MetacardTransformer`	mime-type	`text/xml`
	id	metadata
	shortname (for backwards compatibility)	metadata

Known Issues

None.

4.11.13. Resource Metacard Transformer

The Resource Metacard Transformer retrieves the resource bytes of a metacard by returning the product associated with the metacard.

Installing and Uninstalling

This transformer is installed or uninstalled with the feature `catalog-transformer-resource`.

Configuring

None

Using

Endpoints or other components can retrieve an instance of the Resource Metacard Transformer using its id resource.

Sample Blueprint Reference Snippet

```
<reference id="metacardTransformer" interface="DDF.catalog.transform.MetacardTransformer"
filter="(id=resource)"/>
```

Implementation Details

Service Property	Value	id
resource	shortname	resource
mime-type	application/octet-stream	title

Known Issues

None

4.11.14. Developing a Metacard Transformer

In general, a `MetacardTransformer` is used to transform a `Metacard` into some desired format useful to the end user or as input to another process. Programmatically, a `MetacardTransformer` transforms a `Metacard` into a `BinaryContent` instance, which contains the translated `Metacard` into the desired final format. Metacard transformers can be used through the Catalog Framework `transform` convenience method or requested from the OSGi Service Registry by endpoints or other bundles.

Create a New Metacard Transformer

1. Create a new Java class that implements `DDF.catalog.transform.MetacardTransformer`.
   `public class SampleMetacardTransformer implements DDF.catalog.transform.MetacardTransformer`

2. Implement the transform method.
   `public BinaryContent transform(Metacard metacard, Map<String, Serializable> arguments) throws CatalogTransformerException`

3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: DDF.catalog,DDF.catalog.transform`

4. Create an OSGi descriptor file to communicate with the OSGi Service registry (described in the Working with OSGi section). Export the service to the OSGi registry and declare service properties.

Blueprint descriptor example

```
...
<service ref="[[SampleMetacardTransformer]]" interface=
"DDF.catalog.transform.MetacardTransformer">
    <service-properties>
        <entry key="shortname" value="[[sampletransform]]" />
        <entry key="title" value="[[Sample Metacard Transformer]]" />
        <entry key="description" value="[[A new transformer for metacards.]]" />
    </service-properties>
</service>
...
```

Deploy OSGi Bundle to OSGi runtime.

Variable Descriptions

Table 11. Blueprint Service properties

Key	Description of Value	Example
shortname	(Required) An abbreviation for the return type of the BinaryContent being sent to the user.	atom
title	(Optional) A user-readable title that describes (in greater detail than the shortname) the service.	Atom Entry Transformer Service
description	(Optional) A short, human-readable description that describes the functionality of the service and the output.	This service converts a single metacard xml document to an atom entry element.

4.11.15. Included Query Response Transformers

Query Response transformers convert query responses into other data formats.

Atom Query Response Transformer

The Atom Query Response Transformer transforms a query response into an Atom 1.0 feed. The Atom transformer maps a `QueryResponse` object as described in the Query Result Mapping.

Installing and Uninstalling

The Catalog Transformers application will install this feature when deployed. This transformer's feature, `catalog-transformer-atom`, can be uninstalled or installed.

Configuring

None.

Using

Use this transformer when Atom is the preferred medium of communicating information, such as for feed readers or federation. An integrator could use this with an endpoint to transform query responses into an Atom feed.

For example, clients can use the OpenSearch Endpoint. The client can query with the format option set to the shortname, atom.

Sample OpenSearch query with Atom specified as return format

```
http://localhost:8181/services/catalog/query?q=DDF?format=atom
```

Developers could use this transformer to programmatically transform `QueryResponse` objects on the fly.

Sample Results

Sample Atom Feed from `QueryResponse` *object*

```
 <feed xmlns="http://www.w3.org/2005/Atom" xmlns:os="http://a9.com/-
/spec/opensearch/1.1/">
    <title type="text">Query Response</title>
    <updated>2013-01-31T23:22:37.298Z</updated>
    <id>urn:uuid:a27352c9-f935-45f0-9b8c-5803095164bb</id>
    <link href="#" rel="self" />
    <author>
        <name>Lockheed Martin</name>
    </author>
    <generator version="2.1.0.20130129-1341">DDF123</generator>
    <os:totalResults>1</os:totalResults>
    <os:itemsPerPage>10</os:itemsPerPage>
    <os:startIndex>1</os:startIndex>
    <entry xmlns:relevance="http://a9.com/-/opensearch/extensions/relevance/1.0/"
xmlns:fs="http://a9.com/-/opensearch/extensions/federation/1.0/"
        xmlns:georss="http://www.georss.org/georss">
        <fs:resultSource fs:sourceId="DDF123" />
        <relevance:score>0.19</relevance:score>
        <id>urn:catalog:id:ee7a161e01754b9db1872bfe39d1ea09</id>
        <title type="text">F-15 lands in Libya; Crew Picked Up</title>
        <updated>2013-01-31T23:22:31.648Z</updated>
        <published>2013-01-31T23:22:31.648Z</published>
        <link href=
"http://123.45.67.123:8181/services/catalog/DDF123/ee7a161e01754b9db1872bfe39d1ea09" rel
="alternate" title="View Complete Metacard" />
        <category term="Resource" />
        <georss:where xmlns:gml="http://www.opengis.net/gml">
            <gml:Point>
                <gml:pos>32.8751900768792 13.1874561309814</gml:pos>
            </gml:Point>
        </georss:where>
        <content type="application/xml">
            <ns3:metacard xmlns:ns3="urn:catalog:metacard" xmlns:ns2=
"http://www.w3.org/1999/xlink" xmlns:ns1="http://www.opengis.net/gml"
                xmlns:ns4="http://www.w3.org/2001/SMIL20/" xmlns:ns5=
"http://www.w3.org/2001/SMIL20/Language" ns1:id="4535c53fc8bc4404a1d32a5ce7a29585">
                <ns3:type>DDF.metacard</ns3:type>
                <ns3:source>DDF.distribution</ns3:source>
                <ns3:geometry name="location">
                    <ns3:value>
                        <ns1:Point>
                            <ns1:pos>32.8751900768792 13.1874561309814</ns1:pos>
                        </ns1:Point>
                    </ns3:value>
                </ns3:geometry>
                <ns3:dateTime name="created">
```

```
                    <ns3:value>2013-01-31T16:22:31.648-07:00</ns3:value>
                </ns3:dateTime>
                <ns3:dateTime name="modified">
                    <ns3:value>2013-01-31T16:22:31.648-07:00</ns3:value>
                </ns3:dateTime>
                <ns3:stringxml name="metadata">
                    <ns3:value>
                        <ns6:xml xmlns:ns6="urn:sample:namespace" xmlns=
"urn:sample:namespace">Example description.</ns6:xml>
                    </ns3:value>
                </ns3:stringxml>
                <ns3:string name="metadata-content-type-version">
                    <ns3:value>myVersion</ns3:value>
                </ns3:string>
                <ns3:string name="metadata-content-type">
                    <ns3:value>myType</ns3:value>
                </ns3:string>
                <ns3:string name="title">
                    <ns3:value>Example title</ns3:value>
                </ns3:string>
            </ns3:metacard>
        </content>
    </entry>
</feed>
```

Query Result Mapping

XPath to Atom XML	Value
`/feed/title`	"Query Response"
`/feed/updated`	ISO 8601 dateTime of when the feed was generated
`/feed/id`	Generated UUID URN (http://en.wikipedia.org/wiki/Universally_Unique_Identifier)
`/feed/author/name`	Platform Global Configuration organization
`/feed/generator`	Platform Global Configuration site name
`/feed/generator/@version`	Platform Global Configuration version
`/feed/os:totalResults`	SourceResponse Number of Hits
`/feed/os:itemsPerPage`	Request's Page Size
`/feed/os:startIndex`	Request's Start Index
`/feed/entry/fs:resultSource/@fs:sourceId`	Source Id from which the Result came. `Metacard.getSourceId()`

XPath to Atom XML	Value
/feed/entry/relevance:score	Result's relevance score if applicable. Result.getRelevanceScore()
/feed/entry/id	urn:catalog:id:<Metacard.ID>
/feed/entry/title	Metacard.TITLE
/feed/entry/updated	ISO 8601 dateTime of Metacard.MODIFIED
/feed/entry/published	ISO 8601 dateTime of Metacard.CREATED
/feed/entry/link[@rel='related']	URL to retrieve underlying resource (if applicable and link is available)
/feed/entry/link[@rel='alternate']	Link to alternate view of the Metacard (if a link is available)
/feed/entry/category	Metacard.CONTENT_TYPE
/feed/entry//georss:where	GeoRSS GML of every Metacard attribute with format AttributeFormat.GEOMETRY
/feed/entry/content	Metacard XML generated by DDF.catalog.transform.MetacardTransformer with shortname=xml. If no transformer found, /feed/entry/content/@type will be text and Metacard.ID is displayed .Sample Content with no Metacard Transformation [source,xml] ---- <content type="text">4e1f38d1913b4e93ac622e6c1b258f89</content> ----

XML Query Response Transformer

The XML Query Response Transformer is responsible for translating a query response into an XML formatted document. The metacards element that is generated is an extension of gml:AbstractFeatureCollectionType, which makes the output of this transformer GML 3.1.1 compatible.

Installing and Uninstalling

This transformer comes installed out of the box and is running on start up. To uninstall or install manually, use the catalog-transformer-xml feature.

Configuring

None

Using

Using the OpenSearch Endpoint, for example, query with the format option set to the XML shortname

`xml`.

```
http://localhost:8181/services/catalog/query?q=input?format=xml
```

Example Output

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:metacards xmlns:ns1="http://www.opengis.net/gml" xmlns:ns2=
"http://www.w3.org/1999/xlink" xmlns:ns3="urn:catalog:metacard" xmlns:ns4=
"http://www.w3.org/2001/SMIL20/" xmlns:ns5="http://www.w3.org/2001/SMIL20/Language">
    <ns3:metacard ns1:id="000ba4dd7d974e258845a84966d766eb">
        <ns3:type>DDF.metacard</ns3:type>
        <ns3:source>southwestCatalog1</ns3:source>
        <ns3:dateTime name="created">
          <ns3:value>2013-04-10T15:30:05.702-07:00</ns3:value>
        </ns3:dateTime>
        <ns3:string name="title">
            <ns3:value>Input 1</ns3:value>
        </ns3:string>
    </ns3:metacard>
    <ns3:metacard ns1:id="00c0eb4ba9b74f8b988ef7060e18a6a7">
        <ns3:type>DDF.metacard</ns3:type>
        <ns3:source>southwestCatalog1</ns3:source>
        <ns3:dateTime name="created">
          <ns3:value>2013-04-10T15:30:05.702-07:00</ns3:value>
        </ns3:dateTime>
        <ns3:string name="title">
            <ns3:value>Input 2</ns3:value>
        </ns3:string>
    </ns3:metacard>
</ns3:metacards>
```

Implementation Details

Registered Interface	Service Property	Value
`DDF.catalog.transform.QueryResponseTransformer`	`shortname`	xml
	`description`	Transforms query results into xml
	`title`	View as XML...

Known Issues

None

SearchUI

The SearchUI is a `QueryResponseTransformer` that not only provides results in html format but also provides a convenient, simple querying user interface. It is primarily used as a test tool and verification of configuration. The left pane of the SearchUI contains basic fields to query the Catalog and other Sources. The right pane consists of the results returned from the query.

Installing and Uninstalling

Catalog Transformers App will install this feature when deployed. This transformer's feature, `catalog-transformer-ui`, can be uninstalled or installed.

Configuring

In the Admin Console the SearchUI can be configured under the Catalog HTML Query Response Transformer.

Table 12. Configurable Properties

Title	Property	Type	Description	Default Value	Required
Header	`header`	String	Specifies the header text to be rendered on the SearchUI		yes
Footer	`footer`	String	Specifies the footer text to be rendered on the SearchUI		yes
Template	`template`	String	Specifies the path to the Template	/templates/searchpage.ftl	`yes`
Text Color	`color`	String	Specifies the Text Color of the Header and Footer	yellow	yes
Background Color	`background`	String	Specifies the Background Color of the Header and Footer	green	yes

Using

In order to obtain the SearchUI, a user must use the transformer with an endpoint that queries the Catalog such as the OpenSearch Endpoint. If a distribution is running locally, https://localhost:8993/search/simple should bring up the Simple Search UI. After the page has loaded,

enter the desired search criteria in the appropriate fields. Then click the "Search" button in order to execute the search on the Catalog.

The "Clear" button will reset the query criteria specified.

Query Response Result Mapping

SearchUI Column Title	Catalog Result	Notes
Title	`Metacard.TITLE`	The title may be hyperlinked to view the full Metacard
Source	`Metacard.getSourceId()`	Source where the Metacard was discoved
Location	`Metacard.LOCATION`	Geographical location of the Metacard
Time	`Metacard.CREATED` or `Metacard.EFFECTIVE`	Time received/created
Thumbnail	`Metacard.THUMBNAIL`	No column shown if no results have thumbnail
Resource	`Metacard.RESOURCE_URI`	No column shown if no results have a resource

Search Criteria

The SearchUI allows for querying a Catalog in the following methods:

- Keyword Search - searching with keywords using the grammar of the underlying endpoint/Catalog.
- Temporal Search - searching based on relative or absolute time.
- Spatial search - searching spatially with a Point-Radius or Bounding Box.
- Content Type Search - searching for specific `Metacard.CONTENT_TYPE` values

Known Issues

If the SearchUI results do not provide usable links on the metacard results, verify that a valid host has been entered in the Platform Global Configuration.

4.11.16. Developing a Query Response Transformer

A `QueryResponseTransformer` is used to transform a List of Results from a SourceResponse. Query Response Transformers can be used through the Catalog `transform` convenience method or requested from the OSGi Service Registry by endpoints or other bundles.

4.11.17. Create a New Query Response Transformer

1. Create a new Java class that implements `DDF.catalog.transform.QueryResponseTransformer`.
   ```
   public class SampleResponseTransformer implements DDF.catalog.transform.QueryResponseTransformer
   ```
2. Implement the transform method.
   ```
   public BinaryContent transform(SourceResponse upstreamResponse, Map<String, Serializable> arguments) throws CatalogTransformerException
   ```
3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   ```
   Import-Package: DDF.catalog, DDF.catalog.transform
   ```
4. Create an OSGi descriptor file to communicate with the OSGi Service Registry (described in the Working with OSGi section). Export the service to the OSGi registry and declare service properties.

Blueprint descriptor example

```
...
<service ref="[[SampleResponseTransformer]]" interface=
"DDF.catalog.transform.QueryResponseTransformer">
    <service-properties>
        <entry key="shortname" value="[[sampletransform]]" />
        <entry key="title" value="[[Sample Response Transformer]]" />
        <entry key="description" value="[[A new transformer for response queues.]]" />
    </service-properties>
</service>
...
```

1. Deploy OSGi Bundle to OSGi runtime.

4.11.18. Variable Descriptions

Blueprint Service properties

Key	Description of Value	Example
`shortname`	An abbreviation for the return-type of the BinaryContent being sent to the user.	atom
`title`	A user-readable title that describes (in greater detail than the shortname) the service.	Atom Entry Transformer Service
`description`	A short, human-readable description that describes the functionality of the service and the output.	*This service converts a single metacard xml document to an atom entry element.*

4.11.19. XSLT Transformer

4.11.20. XSLT Transformer Framework

The XSLT Transformer Framework allows developers to create light-weight Query Response Transformers and Metacard Transformers using only a bundle header and XSLT files. The XSLT Transformer Framework registers bundles, following the XSLT Transformer Framework bundle pattern, as new transformer services. The `service-xslt-transformer` feature is part of the DDF core.

Examples

Examples of XSLT Transformers using the XSLT Transformer Framework include `service-atom-transformer` and `service-html-transformer`, found in the services folder of the source code trunk.

4.11.21. Developing an XSLT Transformer

The XSLT Transformer Framework allows developers to create light-weight Query Response Transformers using only a bundle header and XSLT files. The XSLT Transformer Framework registers bundles, following the XSLT Transformer Framework bundle pattern, as new transformer services. The `service-xslt-transformer` feature is part of the DDF core.

Examples

Examples of XSLT Transformers using the XSLT Transformer Framework include `service-atom-transformer` and `service-html-transformer`, found in the services folder of the source code trunk.

Implement an XSLT Transformer

1. Create a new Maven project.
2. Configure the POM to create a bundle using the Maven bundle plugin.
    a. Add the transform output MIME type to the bundle headers.
3. Add XSLT files.

Bundle POM Configuration

Configure the Maven project to create an OSGi bundle using the `maven-bundle-plugin`. Change the DDF-Mime-Type to match the MIME type of the transformer output.

Example POM file

```
...
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.felix</groupId>
            <artifactId>maven-bundle-plugin</artifactId>
            <extensions>true</extensions>
            <configuration>
                <instructions>
                    <DDF-Mime-Type>[[Transform Result MIME Type]]</DDF-Mime-Type>
                    <Bundle-SymbolicName>docs</Bundle-SymbolicName>
                    <Import-Package />
                    <Export-Package />
                </instructions>
            </configuration>
        </plugin>
    </plugins>
</build>
...
```

Including XSLT

The XSLT Transformer Framework will scan for XSLT files inside a bundle. The XSLT file must have a `.xsl` or `.xslt` file in the correct directory location relative to the root of the bundle. The path depends on if the XSLT will act as a Metacard Transformer, Query Response Transformer, or both. The name of the XSLT file will be used as the transformer's shortname.

XSLT File Bundle Path Patterns

```
// Metacard Transformer
<bundle root>
    /OSGI-INF
        /DDF
            /xslt-metacard-transformer
                /<transformer shortname>.[xsl|xslt]

// Query Response Transformer
<bundle root>
    /OSGI-INF
        /DDF
            /xslt-response-queue-transformer
                /<transformer shortname>.[xsl|xslt]
```

The XSLT file has access to metacard or Query Reponse XML data, depending on which folder the XSLT

file is located. The Metacard XML format will depend on the metadata schema used by the Catalog Provider.

For Query Response XSLT Transformers, the available XML data for XSLT transform has the following structure:

Query Response XML

```
<results>
    <metacard>
        <id>[[Metacard ID]]</id>
        <score>[[Relevance score]]</score>
        <distance>[[Distance from query location]]</distance>
        <site>[[Source of result]]</site>
        <type qualifier="type">[[Type]]</type>
        <updated>[[Date last updated]]</updated>
        <geometry>[[WKT geometry]]</geometry>
        <document>
            [[Metacard XML]]
        </document>
    </metacard>
    ...
</results>
```

The XSLT file has access to additional parameters. The `Map<String, Serializable>` arguments from the transform method parameters is merged with the available XSLT parameters.

- Query Response Transformers
  - `grandTotal` - total result count
- Metacard Transformers
  - `id` - metacard ID
  - `siteName` - source ID
  - `services` - list of displayable titles and URLs of available metacard transformers

RSS Example

1. Create a Maven project named `service-rss-transformer`.
2. Add the following to its POM file.

Example RSS POM

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <packaging>bundle</packaging>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <artifactId>services</artifactId>
    <groupId>DDF</groupId>
    <version>[[DDF release version]]</version>
  </parent>
  <groupId>DDF.services</groupId>
  <artifactId>service-rss-transformer</artifactId>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            <DDF-Mime-Type>application/rss+xml</DDF-Mime-Type>
            <Bundle-SymbolicName>docs</Bundle-SymbolicName>
            <Import-Package />
            <Export-Package />
          </instructions>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Line #	Comment
8	Use the current release version.
21	Set the MIME type to the RSS MIME type.

1. Add `service-rss-transformer/src/main/resources/OSGI-INF/DDF/xslt-response-queue-transformer/rss.xsl`. The transformer will be a Query Response Transformer with the shortname `rss` based on the XSL filename and path.

2. Add the following XSL to the new file.

Example RSS XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:gml="http://www.opengis.net/gml" exclude-result-prefixes="xsl gml">

  <xsl:output method="xml" version="1.0" indent="yes" />

  <xsl:param name="grandTotal" />
  <xsl:param name="url" />

  <xsl:template match="/">
    <xsl:apply-templates />
  </xsl:template>

  <xsl:template match="results">
    <rss version="2.0">
      <channel>
      <title>Query Results</title>
      <link><xsl:value-of select="$url" disable-output-escaping="yes" /></link>
      <description>Query Results of <xsl:value-of select="count(//metacard)" /> out of
<xsl:value-of select="$grandTotal" /></description>
      <xsl:for-each select="metacard/document">
        <item>
          <guid>
            <xsl:value-of select="../id" />
          </guid>
          <title>
            <xsl:value-of select="Data/title" />
          </title>
          <link>
            <xsl:value-of select="substring-before($url,'/services')" />
<xsl:text>/services/catalog/</xsl:text><xsl:value-of select="../id" />
<xsl:text>?transform=html</xsl:text>
          </link>
          <description>
            <xsl:value-of select="//description" />
          </description>
          <author>
            <xsl:choose>
              <xsl:when test="Data/creator">
                <xsl:value-of select="Resource/creator//name" />
              </xsl:when>
              <xsl:when test="Data/publisher">
                <xsl:value-of select="Data/publisher//name" />
              </xsl:when>
              <xsl:when test="Data/unknown">
```

```
                <xsl:value-of select="Data/unknown//name" />
              </xsl:when>
            </xsl:choose>
          </author>
          <xsl:if test=".//@posted" >
            <pubDate>
              <xsl:value-of select=".//posted" />
            </pubDate>
          </xsl:if>
          </item>
        </xsl:for-each>
      </channel>
    </rss>
  </xsl:template>
</xsl:stylesheet>
```

Line #	Comment
8-9	Example of using additional parameters and arguments.
15	Example of using the Query Response XML data.
21,27	Example of using the Metacard XML data.

4.12. Extending Federation

Federation provides the capability to extend the DDF enterprise to include Remote Sources, which may include other instances of DDF. The Catalog handles all aspects of federated queries as they are sent to the Catalog Provider and Remote Sources, processed, and the query results are returned. Queries can be scoped to include only the local Catalog Provider (and any Connected Sources), only specific Federated Sources, or the entire enterprise (which includes all local and Remote Sources). If the query is supposed to be federated, the Catalog Framework passes the query to a Federation Strategy, which is responsible for querying each federated source that is specified. The Catalog Framework is also responsible for receiving the query results from each federated source and returning them to the client in the order specified by the particular federation strategy used. After the federation strategy handles the results, the Catalog returns them to the client through the Endpoint. Query results returned from a federated query are a list of metacards. The source ID in each metacard identifies the Source from which the metacard originated.

The Catalog normalizes the incoming query into an OGC Filter format. When the query is disseminated by the Catalog Framework to the sources, each source is responsible for denormalizing the OGC Filter formatted query into the format understood by the external store that the source is acting as a proxy. This normalization/denormalization is what allows any endpoint to interface with any type of source. For example, a query received by the OpenSearch Endpoint can be executed against an OpenSearch Source.

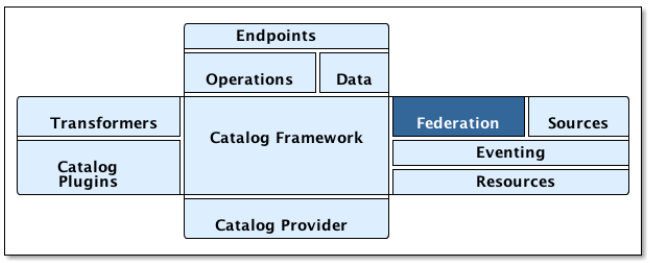


Figure 10. Federation Architecture

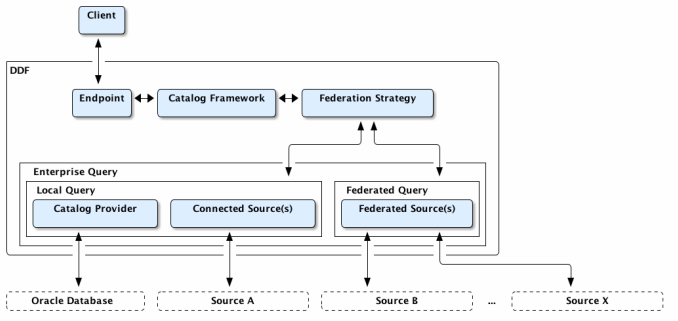


Figure 11. Federation

4.12.1. Federation Strategy

A federation strategy federates a query to all of the Remote Sources in the query's list, processes the results in a unique way, then returns the results to the client. For example, implementations can choose to block until all results return then perform a mass sort or return the results back to the client as soon as they are received back from a Federated Source.

Usage

An endpoint can optionally specify the federation strategy to use when it invokes the query operation. Otherwise, the Catalog provides a default federation strategy that will be used.

Catalog Federation Strategy

The Catalog Federation Strategy is the default federation strategy and is based on sorting metacards by the sorting parameter specified in the federated query.

The possible sorting values are:

- metacard's effective date/time
- temporal data in the query result
- distance data in the query result
- relevance of the query result

The supported sorting orders are ascending and descending.

The default sorting value/order automatically used is relevance descending.

WARNING	The Catalog Federation Strategy expects the results returned from the Source to be sorted based on whatever sorting criteria were specified. If a metadata record in the query results contains null values for the sorting criteria elements, the Catalog Federation Strategy expects that result to come at the end of the result list.

Configuration

The Catalog Federation Strategy configuration can be found in the admin console under **Configuration** **Catalog Federation Strategy**.

Property	Type	Description	Default Value	Required
`maxStartIndex`	Integer	The maximum query offset number (any number from 1 to unlimited). Setting the number too high would allow offset queries that could result in an out of memory error because the DDF will cycle through all records in memory. Things to consider when setting this value are: * How much memory is allocated to the DDF Server * How many sites are being federated with.	50000	yes
`expirationIntervalInMinutes`	Long	Interval that Solr Cache checks for expired documents to remove.	10	yes
`expirationAgeInMinutes`	Long	The number of minutes a document will remain in the cache before it will expire. Default is 7 days.	10080	yes
`url`	String	HTTP URL of Solr Server	https://localhost:8993/solr	yes
`cachingEverything`	Boolean	Cache all results unless configured as native	false	yes

Managed Service PID	`DDF.catalog.federation.impl.CachingFederationStrategy`
Managed Service Factory PID	N/A

4.13. Extending Eventing

The Eventing capability of the Catalog allows endpoints (and thus external users) to create a "standing

query" and be notified when a matching metacard is created, updated, or deleted.

Notably, the Catalog allows event evaluation on both the previous value (if available) and new value of a Metacard when an update occurs.

To better understand why this would be useful, suppose that there has been increased pirating activity off the coast of Somalia. Because of these events, a group of intelligence analysts is interested in determining the reason for the heightened activity and discovering its cause. To do this, analysts need to monitor interesting events occurring in that area. Without DDF Eventing, the analysts would need to repeatedly query for any records of events or intelligence gathered in that area. Analysts would have to monitor changes or anything of interest. However, with DDF Eventing, the analysts can create a subscription indicating criteria for the types of intelligence of interest. In this scenario, analysts could specify interest in metacards added, updated, or deleted that describe data obtained around the coast of Somalia. Through this subscription, DDF will send event notifications back to the team of analysts containing metadata of interest. Furthermore, they could filter the records not only spatially, but by any other criteria that would zero in on the most interesting records. For example, a fishing company that has operated ships peacefully in the same region for a long time may not be interesting. To exclude metadata about that company, analysts may add contextual criteria indicating to return only records containing the keyword "pirate." With the subscription in place, analysts will only be notified of metadata related to the pirating activity, giving them better situational awareness.

The key components of DDF Eventing include:

- Subscription
- Delivery Method
- Event Processor

After reading this section, you will be able to:

- Create new subscriptions
- Register subscriptions
- Perform operations on event notification
- Remove a subscription

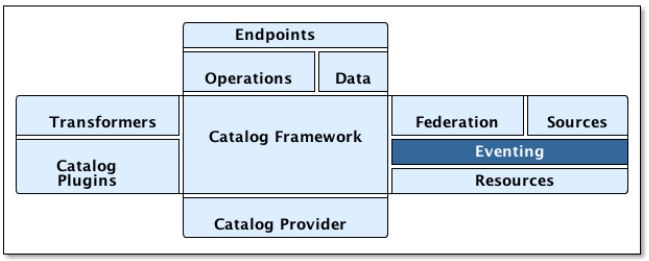


Figure 12. Eventing Architecture

4.13.1. Subscription

Subscriptions represent "standing queries" in the Catalog. Like a query, subscriptions are based on the OGC Filter specification.

Subscription Lifecycle

Creation

- Subscriptions are created directly with the Event Processor or declaratively through use of the Whiteboard Design Pattern.
- The Event Processor will invoke each Pre-Subscription Plugin and, if the subscription is not rejected, the subscription will be activated.

Evaluation

- When a metacard matching the subscription is created, updated, or deleted in any Source, each Pre-Delivery Plugin will be invoked.
- If the delivery is not rejected, the associated Delivery Method callback will be invoked.

Update Evaluation

Notably, the Catalog allows event evaluation on both the previous value (if available) and new value of a Metacard when an update occurs.

Durability

Subscription durability is not provided by the Event Processor. Thus, all subscriptions are transient and will not be recreated in the event of a system restart. It is the responsibility of Endpoints using subscriptions to persist and re-establish the subscription on startup. This decision was made for the sake of simplicity, flexibility, and the inability of the Event Processor to recreate a fully-configured Delivery Method without being overly restrictive.

IMPORTANT	**Subscriptions are not persisted by the Catalog itself.** Subscriptions must be explicitly persisted by an endpoint and are not persisted by the Catalog. The Catalog Framework, or more specifically the Event Processor itself, does not persist subscriptions. Certain endpoints, however, can persist the subscriptions on their own and recreate them on system startup.

Creating a Subscription

Currently, the Catalog reference implementation does not contain a subscription endpoint. Nevertheless, an endpoint that exposes a web service interface to create, update, and delete subscriptions would provide a client's subscription's filtering criteria to be used by Catalog's Event Processor to determine which create, update, and delete events are of interest to the client. The endpoint client also provides the callback URL of the event consumer to be called when an event

matching the subscription's criteria is found. This callback to the event consumer is made by a Delivery Method implementation that the client provides when the subscription is created. Whenever an event occurs in the Catalog matching the subscription, the Delivery Method implementation will be called by the Event Processor. The Delivery Method will, in turn, send the event notification out to the event consumer. As part of the subscription creation process, the Catalog verifies that the event consumer at the specified callback URL is available to receive callbacks. Therefore, the client must ensure the event consumer is running prior to creating the subscription. The Catalog completes the subscription creation by executing any pre-subscription Catalog Plugins, and then registering the subscription with the OSGi Service Registry. The Catalog does not persist subscriptions by default.

Delivery Method

A Delivery Method provides the operation (created, updated, deleted) for how an event's metacard can be delivered.

A Delivery Method is associated with a subscription and contains the callback URL of the event consumer to be notified of events. The Delivery Method encapsulates the operations to be invoked by the Event Processor when an event matches the criteria for the subscription. The Delivery Method's operations are responsible for invoking the corresponding operations on the event consumer associated with the callback URL.

Event Processor

The Event Processor provides an engine that creates, updates, and deletes subscriptions for event notification. These subscriptions optionally specify a filter criteria so that only events of interest to the subscriber are posted for notification.

An internal subscription tracker monitors the OSGi registry, looking for subscriptions to be added (or deleted). When it detects a subscription being added, it informs the Event Processor, which sets up the subscription's filtering and is responsible for posting event notifications to the subscriber when events satisfying their criteria are met.

Event Processing and Notification

As metacards are created, updated, and deleted, the Catalog's Event Processor is invoked (as a post-ingest plugin) for each of these events. TheEvent Processor applies the filter criteria for each registered subscription to each of these ingest events to determine if they match the criteria. If an event matches a subscription's criteria, any pre-delivery plugins that are installed are invoked, the subscription's Delivery Method is retrieved, and its operation corresponding to the type of ingest event is invoked. For example, the DeliveryMethod's `created()` function is called when a metacard is created. The Delivery Method's operations subsequently invoke the corresponding operation in the client's event consumer service, which is specified by the callback URL provided when the Delivery Method was created.

Standard Event Processor

The Standard Event Processor is an implementation of the Event Processor and provides the ability to

create/delete subscriptions. Events are generated by the DDF CatalogFramework as metacards are created/updated/deleted and the Standard Event Processor is called since it is also a Post-Ingest Plugin. The Standard Event Processor checks each event against each subscription's criteria.

When an event matches a subscription's criteria the Standard Event Processor:

- invokes each pre-delivery plugin on the metacard in the event
- invokes the Delivery Method's operation corresponding to the type of event being processed, e.g., created operation for the creation of a metacard

Installing and Uninstalling

The StandardEvent Processor is automatically installed/uninstalled when the Standard Catalog Framework is installed/uninstalled.

Known Issues

The Standard Event processor currently broadcasts federated events and should not. It should only broadcast events that were generated locally, all other events should be dropped.

4.13.2. Fanout Event Processor

The Fanout Event Processor is used when DDF is configured as a fanout proxy. The only difference between the Fanout Event Processor and the Standard Event Processor is that the source ID in the metacard of each event is overridden with the fanout's source ID. This is done to hide the source names of the Remote Sources in the fanout's enterprise. Otherwise, the Fanout Event Processor functions exactly like the Standard Event Processor.

Installing and Uninstalling

The Fanout Event Processor is automatically installed/uninstalled when the Catalog Fanout Framework App is installed/uninstalled.

Known Issues

None

4.13.3. Working with Subscriptions

Creating a Subscription

Using DDF Implementation

If applicable, the implementation of `Subscription` that comes with DDF should be used. It is available at `DDF.catalog.event.impl.SubscriptionImpl` and offers a constructor that takes in all of the necessary objects. Specifically, all that is needed is a `Filter`, `DeliveryMethod`, `Set<String>` of source IDs, and a `boolean` for enterprise.

The following is an example code stub showing how to create a new instance of Subscription using the DDF implementation.

```
// Create a new filter using an imported FilterBuilder
Filter filter = filterBuilder.attribute(Metacard.ANY_TEXT).like().text("*");

// Create a implementation of Delivery Method
DeliveryMethod deliveryMethod = new MyCustomDeliveryMethod();

// Create a set of source ids
// This set is empty as the subscription is not specific to any sources
Set<String> sourceIds = new HashSet<String>();

// Set the isEnterprise boolean value
// This subscription example should notifications from all sources (not just local)
boolean isEnterprise = true;

Subscription subscription = new SubscriptionImpl(filter, deliveryMethod, sourceIds
,isEnterprise);
```

Creating a Custom Implementation

To create a subscription in DDF the developer needs to implement the `DDF.catalog.event.Subscription` interface. This interface extends `org.opengis.filter.Filter` in order to represent the subscription's filter criteria. Furthermore, the `Subscription` interface contains a `DeliveryMethod` implementation.

When implementing `Subscription`, the developer will need to override the methods `accept` and `evaluate` from the `Filter`. The `accept` method allows the visitor pattern to be applied to the `Subscription`. A `FilterVisitor` can be passed into this method in order to process the `Subscription s Filter`. In DDF, this method is used to convert the `Subscription s Filter` into a predicate format that is understood by the Event Processor. The second method inherited from `Filter` is `evaluate`. This method is used to evaluate an object against the `Filter's criteria in order to determine if it matches the criteria.

TIP The functionality of these overridden methods is typically delegated to the `Filter` implementation that the `Subscription` is using.

The developer must also define `getDeliveryMethod`. This class is called when the an event occurs that matches the filter of the subscription.

The other two methods required because `Subscription` implements `Federatable` are `isEnterprise` and `getSourceIds`, which indicate that the subscription should watch for events occurring on all sources in the enterprise or on specified sources.

The following is an implementation stub of `Subscription` that comes with DDF and is available

at DDF.catalog.event.impl.SubscriptionImpl.

SubscriptionImpl

```
public class SubscriptionImpl implements Subscription {
    private Filter filter;

    private DeliveryMethod dm;

    private Set<String> sourceIds;

    private boolean enterprise;

    public SubscriptionImpl(Filter filter, DeliveryMethod dm, Set<String> sourceIds,
            boolean enterprise) {
        this.filter = filter;
        this.dm = dm;
        this.sourceIds = sourceIds;
        this.enterprise = enterprise;
    }

    @Override
    public boolean evaluate(Object object) {
        return filter.evaluate(object);
    }

    @Override
    public Object accept(FilterVisitor visitor, Object extraData) {
        return filter.accept(visitor, extraData);
    }

    @Override
    public Set<String> getSourceIds() {
        return sourceIds;
    }

    @Override
    public boolean isEnterprise() {
        return enterprise;
    }

    @Override
    public DeliveryMethod getDeliveryMethod() {
        return dm;
    }
}
```

4.13.4. Registering a Subscription

Once a `Subscription` is created, it needs to be registered in the OSGi Service Registry as a `DDF.catalog.event.Subscription` service. This is necessary for the `Subscription` to be discovered by the Event Processor. Typically, this is done in code after the `Subscription` is instantiated. When the `Subscription` is registered, a unique ID will need to be specified using the key `subscription-id`. This will be used to delete the `Subscription` from the OSGi Service Registry. Furthermore, the `ServiceRegistration`, which is the return value from registering a `Subscription`, should be monitored in order to remove the `Subscription` later. The following code shows how to correctly register a `Subscription` implementation in the registry using the above `SubscriptionImpl` for clarity:

Registering a Subscription

```
// Map to keep track of registered Subscriptions.  Used for unregistering Subscriptions.
Map<String, ServiceRegistration<Subscription>> subscriptions = new HashMap<String,
ServiceRegistration<Subscription>>();

// New subscription using the DDF Implementation of subscription
Subscription subscription = new SubscriptionImpl(filter, deliveryMethod, sourceIds
,isEnterprise);

// Specify the subscription-id to uniquely identify the Subscription
String subscriptionId = "0123456789abcdef0123456789abcdef";
Dictionary<String, String> properties = new Hashtable<String, String>();
properties.put("subscription-id", subscriptionId);

// Service registration requires an instance of the OSGi bundle context
// Register subscription and keep track of the service registration
ServiceRegistration<Subscription> serviceRegistration = context.registerService(DDF
.catalog.event.Subscription.class, subscription, properties );
subscriptions.put(subscriptionId, serviceRegistration);
```

4.13.5. Creating a Delivery Method

The Event Processor obtains the subscription's `DeliveryMethod` and invokes one of its four methods when an event occurs. The `DeliveryMethod` then handles that invocation and communicates an event to a specified consumer service outside of DDF.

The Event Processor calls the `DeliveryMethod  s`created` method when a new metacard matching the filter criteria is added to the Catalog. It calls the `deleted` method when a metacard that matched the filter criteria is removed from the Catalog. `updatedHit` is called when a metacard is updated and the new metacard matches the subscription. `updatedMiss` is different in that it is only called if the old metacard matched the filter but the new metacard no longer does. An example of this would be if the filter contains spatial criteria consisting of Arizona. If a plane is flying over Arizona, the Event Processor will repeatedly call `updatedHit` as the plane flies from one side to the other while updating its position in the Catalog. This happens because the updated records continually match the specified

criteria. If the plane crosses into New Mexico, the previous metacard will have matched the filter, but the new metacard will not. Thus, `updatedMiss` gets called.

The following is an implementation stub for `DeliveryMethod`:

DeliveryMethodImpl

```
public class DeliveryMethodImpl implements DeliveryMethod {

    @Override
    public void created(Metacard newMetacard) {
        // Perform custom code on create
    }

    @Override
    public void updatedHit(Metacard newMetacard, Metacard oldMetacard) {
// Perform custom code on update (where both new and old metacards matched filter)
    }

    @Override
    public void updatedMiss(Metacard newMetacard, Metacard oldMetacard) {
// Perform custom code on update (where one of the two metacards did not match the
filter)
    }

    @Override
    public void deleted(Metacard oldMetacard) {
     // Perform custom code on delete
    }
}
```

4.13.6. Deleting a Subscription

To remove a subscription from DDF, the subscription ID is required. Once this is provided, the `ServiceRegistration` for the indicated `Subscription` should be obtained from the `Subscriptions` Map. Then the `Subscription` can be removed by unregistering the service. The following code demonstrates how this is done:

Delete Subscription

```
String subscriptionId = "0123456789abcdef0123456789abcdef";

//Obtain service registration from subscriptions Map based on subscription ID
ServiceRegistration<Subscription> sr = subscriptions.get(subscriptionId);

//Unregister Subscription from OSGi Service Registry
sr.unregister();

//Remove Subscription from Map keeping track of registered Subscriptions.
subscriptions.remove(subscriptionId);
```

4.14. Extending Resource Components

Resource components are used when working with resources, i.e., the data that is represented by the cataloged metadata.

A resource is a URI-addressable entity that is represented by a metacard. Resources may also be known as products or data.

Resources may exist either locally or on a remote data store.

Examples of resources include:

- NITF image
- MPEG video
- Live video stream
- Audio recording
- Document

A resource object in DDF contains an `InputStream` with the binary data of the resource. It describes that resource with a name, which could be a file name, URI, or another identifier. It also contains a mime type or content type that a client can use to interpret the binary data.

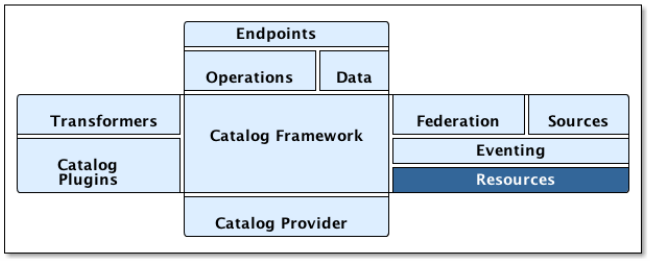


Figure 13. Resources Architecture

4.14.1. Resource Readers

A resource reader retrieves resources associated with metacards via URIs. Each resource reader must know how to interpret the resource's URI and how to interact with the data store to retrieve the resource.

There can be multiple resource readers in a Catalog instance. The `Catalog Framework` selects the appropriate resource reader based on the scheme of the resource's URI.

In order to make a resource reader available to the Catalog Framework, it must be exported to the OSGi Service Registry as a `DDF.catalog.resource.ResourceReader`.

URL Resource Reader

The `URLResourceReader` is an implementation of `ResourceReader` which is included in the DDF Catalog. It obtains a resource given an http, https, or file-based URL. The `URLResourceReader` will connect to the provided Resource URL and read the resource's bytes into an `InputStream`.

WARNING

When a resource linked using a file-based URL is in the product cache, the `URLResourceReader`'s rootResourceDirectories is not checked when downloading the product. It is downloaded from the product cache which bypasses the `URLResourceReader`. For example, if path `/my/valid/path` is configured in the `URLResourceReader`'s rootResourceDirectories and one downloads the product with resource-uri `file:///my/valid/path/product.txt` and then one removes `/my/valid/path` from the `URLResourceReader`'s `rootResourceDirectories` configuration, the product will still be accessible via the product cache.

Installing and Uninstalling

`URLResourceReader` is installed by default with the DDF Catalog.

Configuring

Configurable Properties

URL Resource Reader

Property	Type	Description	Default Value	Required
`rootResourceDirectories`	String array	Specifies the only directories the `URLResourceReader` has access to when attempting to download resources linked using file-based URLs (i.e. the metacard attribute resource-uri uses the file URI scheme). This property is used to restrict the `URLResourceReader`'s access to the file system. The `URLResourceReader` can be given full access to the file system by setting the `rootResourceDirectories` property to the root directory (e.g. `/`), but this configuration is not recommended.	<DDF.home>/data/products	yes

Using

`URLResourceReader` will be used by the Catalog Framework to obtain a resource whose metacard is cataloged in the local data store. This particular `ResourceReader` will be chosen by the `CatalogFramework` if the requested resource's URL has a protocol of `http`, `https`, or `file`.

For example, requesting a resource with the following URL will make the Catalog Framework invoke the `URLResourceReader` to retrieve the product.

Example

```
file:///home/users/DDF_user/data/example.txt
```

If a resource was requested with the URL `udp://123.45.67.89:80/SampleResourceStream`, the `URLResourceReader` would *not* be invoked.

Implementation Details

Supported Schemes:

- http
- https
- file

NOTE	If a file-based URL is passed to the `URLResourceReader`, that file path needs to be accessible by the DDF instance.

Known Issues

None

4.14.2. Developing a Resource Reader

A `ResourceReader` is a class that retrieves a resource or product from a native/external source and returns it to DDF. A simple example is that of a File `ResourceReader`. It takes a file from the local file system and passes it back to DDF. New implementations can be created in order to support obtaining Resources from various Resource data stores.

Create a New ResourceReader

Complete the following procedure to create a `ResourceReader`.

1. Create a Java class that implements the `DDF.catalog.resource.ResourceReader` interface.
2. Deploy the OSGi bundled packaged service to the DDF run-time.

Implementing the ResourceReader Interface

```
public class TestResourceReader implements DDF.catalog.resource.ResourceReader
```

`ResourceReader` has a couple of key methods where most of the work is performed.

NOTE	**URI** It is recommended to become familiar with the Java API URI class in order to properly build a `ResourceReader`. Furthermore, a URI should be used according to its specification.

retrieveResource

```
public ResourceResponse retrieveResource( URI uri, Map<String, Serializable> arguments
)throws IOException, ResourceNotFoundException, ResourceNotSupportedException;
```

This method is the main entry to the `ResourceReader`. It is used to retrieve a `Resource` and send it back to the caller (generally the `CatalogFramework`). Information needed to obtain the entry is contained in the `URI` reference. The URI Scheme will need to match a scheme specified in the `getSupportedSchemes` method. This is how the CatalogFramework determines which `ResourceReader` implementation to use. If there are multiple `ResourceReaders` supporting the same scheme, these `ResourceReaders` will be invoked iteratively. Invocation of the `ResourceReaders` stops once one of them returns a `Resource`.

Arguments are also passed in. These can be used by the `ResourceReader` to perform additional operations on the resource.

An example of how `URLResourceReader` (located in the source code at `/trunk/DDF/catalog/resource/URLResourceReader.java`) implements the `getResource` method. This `ResourceReader` simply reads a file from a URI.

NOTE	The `Map<String, Serializable> arguments` parameter is passed in to support any options or additional information associated with retrieving the resource.

Implement `retrieveResource()`

1. Define supported schemes (e.g., file, http, etc.).
2. Check if the incoming URI matches a supported scheme. If it does not, throw `ResourceNotSupportedException`.

Example:

```
if ( !uri.getScheme().equals("http") )
 {
   throw new ResourceNotSupportedException("Unsupported scheme received, was expecting
http")
 }
```

1. Implement the business logic.
2. For example, the `URLResourceReader` will obtain the resource through a connection:

```
URL url = uri.toURL();
URLConnection conn = url.openConnection();
String mimeType = conn.getContentType();
if ( mimeType == null ) {
    mimeType = URLConnection.guessContentTypeFromName( url.getFile() );
}
InputStream is = conn.getInputStream();
```

NOTE	The `Resource` needs to be accessible from the DDF installation (see the rootResourceDirectories property of the `URLResourceReader`). This includes being able to find a file locally or reach out to a remote URI. This may require Internet access, and DDF may need to be configured to use a proxy (`http.proxyHost` and `http.proxyPort` can be added to the system properties on the command line script).

1. Return `Resource` in `ResourceResponse`.

For example:

```
return ResourceResponseImpl( new ResourceImpl( new BufferedInputStream( is ), new
MimeType( mimeType ), url.getFile() ) );
```

If the Resource cannot be found, throw a `ResourceNotFoundException`.

`getSupportedSchemes`

```
public Set<String> getSupportedSchemes();
```

This method lets the `ResourceReader` inform the CatalogFramework about the type of URI scheme that it accepts and should be passed. For single-use ResourceReaders (like a URLResourceReader), there may be only one scheme that it can accept while others may understand more than one. A ResourceReader must, at minimum, accept one qualifier. As mentioned before, this method is used by the `CatalogFramework` to determine which `ResourceReader` to invoke.

NOTE	**`ResourceReader` extends `Describable`** Additionally, there are other methods that are used to uniquely describe a `ResourceReader`. The `describe` methods are straight-forward and can be implemented with guidance from the Javadoc.

Export to OSGi Service Registry

In order for the `ResourceReader` to be used by the `CatalogFramework`, it should be exported to the OSGi Service Registry as a `DDF.catalog.resource.ResourceReader`.

See the XML below for an example:

Blueprint example

```
<bean id="[[customResourceReaderId]]" class=
"[[example.resource.reader.impl.CustomResourceReader]]" />
<service ref="[[customResourceReaderId]]" interface="DDF.catalog.source.ResourceReader"
/>
```

4.14.3. Resource Writers

A resource writer stores a resource and produces a URI that can be used to retrieve the resource at a later time. The resource URI uniquely locates and identifies the resource. Resource writers can interact with an underlying data store and store the resource in the proper place. Each implementation can do this differently, providing flexibility in the data stores used to persist the resources.

Examples

The Catalog reference implementation currently does not include any resource writers out of the box.

4.14.4. Developing a Resource Writer

A `ResourceWriter` is an object used to store or delete a `Resource`. `ResourceWriter` objects should be registered within the OSGi Service Registry, so clients can retrieve an instance when clients need to store a `Resource`.

Create a New `ResourceWriter`

Complete the following procedure to create a `ResourceWriter`.

1. Create a Java class that implements the `DDF.catalog.resource.ResourceWriter` interface.

ResourceWriter Implementation Skeleton

```
import java.io.IOException;
import java.net.URI;
import java.util.Map;
import DDF.catalog.resource.Resource;
import DDF.catalog.resource.ResourceNotFoundException;
import DDF.catalog.resource.ResourceNotSupportedException;
import DDF.catalog.resource.ResourceWriter;

public class SampleResourceWriter implements ResourceWriter {

@Override
public void deleteResource(URI uri, Map<String, Object> arguments) throws
ResourceNotFoundException, IOException {
// WRITE IMPLEMENTATION
}

@Override
public URI storeResource(Resource resource, Map<String, Object> arguments)throws
ResourceNotSupportedException, IOException {
// WRITE IMPLEMENTATION
return null;
}

@Override
public URI storeResource(Resource resource, String id, Map<String, Object> arguments)
throws ResourceNotSupportedException, IOException {
// WRITE IMPLEMENTATION
return null;
}

}
```

1. Register the implementation as a Service in the OSGi Service Registry.

Blueprint Service Registration Example

```
...
<service ref="[[ResourceWriterReference]]" interface="
DDF.catalog.resource.ResourceWriter" />
...
```

1. Deploy the OSGi bundled packaged service to the DDF run-time (Refer to the Working with OSGi - Bundles section.)

TIP	**ResourceWriter Javadoc** Refer to the DDF Catalog API Javadoc for more information about the methods required for implementing the interface.

4.14.5. Developing a Registry Client

Registry Clients create Federated Sources using the OSGi Configuration Admin. Developers should reference an individual `Source  s (Federated, Connected, or Catalog Provider) documentation for the Configuration properties (such as a Factory PID, addresses, intervals, etc) necessary to establish that `Source` in the framework.

Example

Creating a Source Configuration

```
org.osgi.service.cm.ConfigurationAdmin configurationAdmin = getConfigurationAdmin() ;
org.osgi.service.cm.Configuration currentConfiguration = configurationAdmin
.createFactoryConfiguration(getFactoryPid(), null);
Dictionary properties = new Dictionary() ;
properties.put(QUERY_ADDRESS_PROPERTY,queryAddress);
currentConfiguration.update( properties );
```

Note that the `QUERY_ADDRESS_PROPERTY` is specific to this Configuration and might not be required for every `Source`. The properties necessary for creating a Configuration are different for every `Source`.

4.14.6. Working with Resources

Metacards and Resources

Metacards are used to describe a resource through metadata. This metadata includes the time the resource was created, the location where the resource was created, etc. A DDF `Metacard` contains the `getResourceUri` method, which is used to locate and retrieve its corresponding resource.

Retrieve Resource

When a client attempts to retrieve a resource, it must provide a metacard ID or URI corresponding to a

unique resource. As mentioned above, the resource URI is obtained from a `Metacard`'s `getResourceUri` method. The `CatalogFramework` has three methods that can be used by clients to obtain a resource: `getEnterpriseResource`, `getResource`, and `getLocalResource`. The `getEnterpriseResource` method invokes the `retrieveResource` method on a local `ResourceReader` as well as all the `Federated` and `Connected` Sources inthe DDF enterprise. The second method, `getResource`, takes in a source ID as a parameter and only invokes `retrieveResource` on the specified `Source`. The third method invokes `retrieveResource` on a local `ResourceReader`.

The parameter for each of these methods in the `CatalogFramework` is a `ResourceRequest`. DDF includes two implementations of `ResourceRequest`: `ResourceRequestById` and `ResourceRequestByProductUri`. Since these implementations extend `OperationImpl`, they can pass a `Map` of generic properties through the `CatalogFramework` to customize how the resource request is carried out. One example of this is explained in the Options section below. The following is a basic example of how to create a `ResourceRequest` and invoke the `CatalogFramework` resource retrieval methods to process the request.

Retrieve Resource Example

```
Map<String, Serializable> properties = new HashMap<String, Serializable>();
properties.put("PropertyKey1", "propertyA"); //properties to customize Resource retrieval
ResourceRequestById resourceRequest = new ResourceRequestById(
"0123456789abcdef0123456789abcdef", properties); //object containing ID of Resource to be
retrieved
String sourceName = "LOCAL_SOURCE"; //the Source ID or name of the local Catalog or a
Federated Source
ResourceResponse resourceResponse; //object containing the retrieved Resource and the
request that was made to get it.
resourceResponse = catalogFramework.getResource(resourceRequest, sourceName); //Source-
based retrieve Resource request
Resource resource = resourceResponse.getResource(); //actual Resource object containing
InputStream, mime type, and Resource name
```

`DDF.catalog.resource.ResourceReader` instances can be discovered via the OSGi Service Registry. The system can contain multiple `ResourceReaders`. The `CatalogFramework` determines which one to call based on the scheme of the resource's URI and what schemes the `ResourceReader` supports. The supported schemes are obtained by a `ResourceReader`'s `getSupportedSchemes` method. As an example, one `ResourceReader` may know how to handle file-based URIs with the scheme `file`, whereas another `ResourceReader` may support HTTP-based URIs with the scheme `http`.

The `ResourceReader` or `Source` is responsible for locating the resource, reading its bytes, adding the binary data to a `Resource` implementation, then returning that `Resource` in a `ResourceResponse`. The `ResourceReader` or `Source` is also responsible for determining the `Resource`'s name and mime type, which it sends back in the `Resource` implementation.

Options

Options can be specified on a retrieve resource request made through any of the supporting endpoint.

To specify an option for a retrieve resource request, the endpoint needs to first instantiate a `ResourceRequestByProductUri` or a `ResourceRequestById`. Both of these `ResourceRequest` implementations allow a `Map` of properties to be specified. Put the specified option into the `Map` under the key `RESOURCE_OPTION`.

Retrieve Resource with Options

```
Map<String, Serializable> properties = new HashMap<String, Serializable>();
properties.put("RESOURCE_OPTION", "OptionA");
ResourceRequestById resourceRequest = new ResourceRequestById(
"0123456789abcdef0123456789abcdef", properties);
```

Depending on the support that the `ResourceReader` or `Source` provides for options, the `properties``Map` will be checked for the `RESOURCE_OPTION` entry. If that entry is found, the option will be handled; however, the `ResourceReader` or `Source` supports options. If the `ResourceReader` or `Source` does not support options, that entry will be ignored.

A new `ResourceReader` or `Source` implementation can be created to support options in a way that is most appropriate. Since the option is passed through the catalog framework as a property, the `ResourceReader` or `Source` will have access to that option as long as the endpoint supports options.

Store Resource

Resources are saved using a `ResourceWriter`. `DDF.catalog.resource.ResourceWriter` instances can be discovered via the OSGi Service Registry. Once retrieved, the `ResourceWriter` instance provides clients a way to store resources and get a corresponding URI that can be used to subsequently retrieve the resource via a `ResourceReader`. Simply invoke either of the `storeResource` methods with a resource and any potential arguments. The `ResourceWriter` implementation is responsible for determining where the resource is saved and how it is saved. This allows flexibility for a resource to be saved in any one of a variety of data stores or file systems. The following is an example of how to use a generic implementation of `ResourceWriter`.

```
InputStream inputStream = <Video_Input_Stream>; //InputStream of raw Resource data
MimeType mimeType = new MimeType("video/mpeg"); //Mime Type or content type of Resource
String name = "Facility_Video";  //Descriptive Resource name
Resource resource = new ResourceImpl(inputStream, mimeType, name);
Map<String, Object> optionalArguments = new HashMap<String, Object>();
ResourceWriter writer = new ResourceWriterImpl();
URI resourceUri; //URI that can be used to retrieve Resource
resourceUri = writer.storeResource(resource, optionalArguments); //Null can be passed in
here
```

BinaryContent

`BinaryContent` is an object used as a container to store translated or transformed DDF components. `Resource` extends `BinaryContent` andincludes a `getName` method. ` BinaryContent` has methods to get the `InputStream`, `byte` array, MIME type, and size of the represented binary data. An implementation of `BinaryContent` (`BinaryContentImpl`) can be found in the Catalog API in the `DDF.catalog.data` package.

Additional Information

- URI on Wikipedia (http://en.wikipedia.org/wiki/Uniform_resource_identifier)
- URI Javadoc (http://docs.oracle.com/javase/6/docs/api/java/net/URI.html)

5. Extending DDF Platform

Version: 2.9.1

This section supports developers creating extensions of the existing framework.

5.1. Whitelist

The following packages have been exported by the DDF Platform application and are approved for use by third parties:

- `ddf.action`
- `ddf.security`
- `ddf.security.assertion`
- `ddf.security.common.audit`
- `ddf.security.http`
- `ddf.security.permission`
- `ddf.security.policy.extension`
- `ddf.security.principal`
- `ddf.security.samlp`
- `ddf.security.service`
- `ddf.security.settings`
- `ddf.security.sts.client.configuration`
- `ddf.security.ws.policy`
- `ddf.security.ws.proxy`
- `org.codice.ddf.branding`
- `org.codice.ddf.configuration`
- `org.codice.ddf.configuration.admin`
- `org.codice.ddf.configuration.migration`
- `org.codice.ddf.configuration.persistence`
- `org.codice.ddf.configuration.persistence.felix`
- `org.codice.ddf.configuration.status`
- `org.codice.ddf.notifications.store`

- `org.codice.ddf.parser`
- `org.codice.ddf.parser.xml`
- `org.codice.ddf.platform.error.handler`
- `org.codice.ddf.platform.util`

WARNING	The Platform Application includes other third party packages such as Apache CXF and Apache Camel. These are available for use by third party developers but their versions can change at any time with future releases of the Platform Application.

5.2. Developing Action Components (Action Framework)

The Action Framework was designed as a way to limit dependencies between applications (apps) in a system. For instance, a feature in an app, such as a Atom feed generator, might want to include an external link as part of its feed's entries. That feature does not have to be coupled to a REST endpoint to work, nor does it have to depend on a specific implementation to get a link. In reality, the feature does not identify how the link is generated, but it does identify whether link works or does not work when retrieving the intended entry's metadata. \Instead of creating its own mechanism or adding an unrelated feature, it could use the Action Framework to query out in the OSGi container for any service that can provide a link. This does two things: it allows the feature to be independent of implementations, and it encourages reuse of common services.

The Action Framework consists of two major Java interfaces in its API:

1. `ddf.action.Action`
2. `ddf.action.ActionProvider`

5.2.1. Usage

To provide a service, such as a link to a record, the `ActionProvider` interface should be implemented. An `ActionProvider` essentially provides a List of `Action`s when given input that it can recognize and handle. For instance, if a REST endpoint ActionProvider was given a metacard, it could provide a link based on the metacard s ID. An Action Provider performs an action when given a subject that it understands. If it does not understand the subject or does not know how to handle the given input, it will return `Collections.emptyList()`. An Action Provider is required to have an ActionProvider id. The Action Provider must register itself in the OSGi Service Registry with the `ddf.action.ActionProvider` interface and must also have a service property value for `id`. An action is a URL that, when invoked, provides a resource or executes intended business logic.

5.2.2. Naming Convention

For each Action, a title and description should be provided to describe what the action does. The recommended naming convention is to use the verb 'Get' when retrieving a portion of the metacard, such as the metadata or thumbnail, or when you are downloading the product. The verb 'Export' or expression 'Export as' is recommended when the metacard is bring exported in a different format or

presented after going some transformation.

5.2.3. Taxonomy

An Action Provider registers an `id` as a service property in the OGSi Service Registry based on the type of service or action that is provided. Regardless of implementation, if more than one Action Provider provides the same service, such as providing a URL to a thumbnail for a given metacard, they must both register under the same `id`. Therefore, Action Provider implementers must follow an Action Taxonomy.

The following is a sample taxonomy:

1. `catalog.data.metacard` shall be the grouping that represents Actions on a Catalog metacard.
    a. `catalog.data.metacard.view`
    b. `catalog.data.metacard.thumbnail`
    c. `catalog.data.metacard.html`
    d. `catalog.data.metacard.resource`
    e. `catalog.data.metacard.metadata`

Action ID Service Descriptions

ID	Required Action	Naming Convention
`catalog.data.metacard.view`	Provides a valid URL to view all of a metacard data. Format of data is not specified; i.e. the representation can be in XML, JSON, or other.	Export as ...
`catalog.data.metacard.thumbnail`	Provides a valid URL to the bytes of a thumbnail (`Metacard.THUMBNAIL`) with MIME type image/jpeg.	Get Thumbnail
`catalog.data.metacard.html`	Provides a valid URL that, when invoked, provides an HTML representation of the metacard.	Export as ...
`catalog.data.metacard.resource`	Provides a valid URL that, when invoked, provides the underlying resource of the metacard.	Get Resource
`catalog.data.metacard.metadata`	Provides a valid URL to the XML metadata in the metacard (`Metacard.METADATA`).	Get Metadata

5.3. Developing Migratables

The `Migratable` API provides a mechanism for bundles to handle exporting data required to clone a DDF system. The migration process is meant to be flexible, so an implementation of `org.codice.ddf.migration.Migratable` can handle exporting data for a single bundle or groups of

bundles such as applications. For example, the `org.codice.ddf.platform.migratable.impl.PlatformMigratable` handles exporting core system files for the DDF Platform Application. Exporting configurations stored in `org.osgi.service.cm.ConfigurationAdmin` does not need to be handled by implementations of `org.codice.ddf.migration.Migratable` as all `ConfigurationAdmin` configurations are exported by `org.codice.ddf.configuration.admin.ConfigurationAminMigration`.

The Migratable API includes:

1. `org.codice.ddf.migration.Migratable`
2. `org.codice.ddf.migration.AbstractMigratable`
3. `org.codice.ddf.migration.MigrationException`
4. `org.codice.ddf.migration.MigrationMetadata`
5. `org.codice.ddf.migration.MigrationWarning`

5.3.1. Usage

The `org.codice.ddf.migration.Migratable` interface defines these methods:

The `exportPath` in `export(Path exportPath)` is the path where all of the exportable data is copied. It is provided via an argument to the `migration:export` console command or via the Export Dialog in the Admin Console. The default value is `<DISTRIBUTION HOME>/etc/exported`. It is the responsibility of a `Migratable` to prevent naming collisions upon export. For example, if a `Migratable` writes files for its export, it must namespace the files. The `getDescription()` operation returns a short description of the type of data exported by the `Migratable`. The `isOptional()` operation returns whether the exported data for the `Migratable` is optional or required. The description and optional flag are for display purposes in the Admin Console.

A `org.codice.ddf.migration.MigrationException` should be thrown when an unrecoverable exception occurs that prevents required data from exporting. The exception message is displayed to the admin.

A `org.codice.ddf.migration.MigrationWarning` should be used when a `Migratable` wants to warn an admin that certain aspects of the export may cause problems upon import. For example, if an absolute path is encountered, that path may not exist on the target system and cause the installation to fail. All migration warnings are displayed to the admin.

In order to create a `Migratable` for a module of the system, the `org.codice.ddf.migration.Migratable` interface must be implemented and the implementation must be registered under the `org.codice.ddf.migration.Migratable` interface as an OSGI service in the OSGI service registry. Creating an OSGI service allows for the `org.codice.ddf.configuration.migration.ConfigurationMigrationManager` to lookup all implementations of `org.codice.ddf.migration.Migratable` and command them to export.

The abstract base class `org.codice.ddf.migration.AbstractMigratable` in the `platform-migratable-api` implements common boilerplate code required when implementing

`org.codice.ddf.migration.Migratable` and should be extended when creating a `org.codice.ddf.migration.Migratable`.

5.4. Do Not Use FileBackedOutputStream

It is recommended to avoid using `com.google.common.io.FileBackedOutputStream` (FBOS). FBOS can create temporary files that are not automatically removed when FBOS is closed. Use `org.codice.ddf.platform.util.TemporaryFileBackedOutputStream` (TFBOS). TFBOS provides the same method calls as FBOS, but will remove temporary files when it is closed.

6. Extending DDF Security

Version: 2.9.1

This section supports developers creating extensions of the existing framework.

6.1. Whitelist

The following packages have been exported by the DDF Security application and are approved for use by third parties:

- `ddf.security.assertion.impl`
- `ddf.security.common.util`
- `ddf.security.encryption`
- `ddf.security.expansion`
- `ddf.security.http.impl`
- `ddf.security.impl`
- `ddf.security.pdp.realm`
- `ddf.security.realm.sts`
- `ddf.security.samlp.impl`
- `ddf.security.service.impl`
- `ddf.security.soap.impl`
- `ddf.security.sts`
- `ddf.security.ws.policy.impl`
- `org.apache.cxf.sts.cache`
- `org.apache.cxf.sts.claims`
- `org.apache.cxf.sts.event.map`
- `org.apache.cxf.sts.event`
- `org.apache.cxf.sts.interceptor`
- `org.apache.cxf.sts.operation`
- `org.apache.cxf.sts.provider`
- `org.apache.cxf.sts.request`
- `org.apache.cxf.sts.service`

- org.apache.cxf.sts.token.canceller
- org.apache.cxf.sts.token.delegation
- org.apache.cxf.sts.token.provider
- org.apache.cxf.sts.token.realm
- org.apache.cxf.sts.token.renewer
- org.apache.cxf.sts.token.validator
- org.apache.cxf.sts
- org.codice.ddf.security.certificate.generator
- org.codice.ddf.security.certificate.keystore.editor
- org.codice.ddf.security.common
- org.codice.ddf.security.filter.authorization
- org.codice.ddf.security.filter.login
- org.codice.ddf.security.filter.websso
- org.codice.ddf.security.handler.api
- org.codice.ddf.security.handler.basic
- org.codice.ddf.security.handler.guest.configuration
- org.codice.ddf.security.handler.guest
- org.codice.ddf.security.handler.pki
- org.codice.ddf.security.handler.saml
- org.codice.ddf.security.interceptor
- org.codice.ddf.security.interceptor
- org.codice.ddf.security.policy.context.attributes
- org.codice.ddf.security.policy.context.impl
- org.codice.ddf.security.policy.context
- org.codice.ddf.security.servlet.logout
- org.codice.ddf.security.validator.username

6.2. Developing Token Validators

Token validators are used by the Security Token Service (STS) to validate incoming token requests. The `TokenValidator` CXF interface must be implemented by any custom token validator class. The

`canHandleToken` and `validateToken` methods must be overridden. The `canHandleToken` method should return true or false based on the `ValueType` value of the token that the validator is associated with. The validator may be able to handle any number of different tokens that you specify. The `validateToken` method returns a `TokenValidatorResponse` object that contains the `Principal` of the identity being validated and also validates the `ReceivedToken` object that was collected from the RST (`RequestSecurityToken`) message.

7. Extending DDF Solr

Version: 2.9.1

7.1. Whitelist

The following packages have been exported by the DDF Solr application and are approved for use by third parties:

None.

8. Extending DDF Spatial

Version: 2.9.1

8.1. Whitelist

The following packages have been exported by the DDF Spatial Application and are approved for use by third parties:

- net.opengis.cat.csw.v_2_0_2.dc.elements
- net.opengis.cat.csw.v_2_0_2.dc.terms
- net.opengis.cat.csw.v_2_0_2
- net.opengis.filter.v_1_1_0
- net.opengis.gml.v_3_1_1
- net.opengis.ows.v_1_0_0
- org.codice.ddf.spatial.geocoder.geonames
- org.codice.ddf.spatial.geocoder
- org.codice.ddf.spatial.geocoding.context
- org.codice.ddf.spatial.geocoding
- org.codice.ddf.spatial.kml.endpoint
- org.codice.ddf.spatial.kml.transformer
- org.codice.ddf.spatial.ogc.catalog.resource.impl
- org.codice.ddf.spatial.ogc.catalog
- org.codice.ddf.spatial.ogc.wfs.catalog.converter
- org.codice.ddf.spatial.ogc.wfs.catalog.mapper
- org.codice.ddf.spatial.ogc.wfs.v1_0_0.catalog.converter
- org.codice.ddf.spatial.ogc.wfs.v2_0_0.catalog.converter

9. Extending DDF Search UI

Version: 2.9.1

This section supports developers creating extensions of the existing framework.

9.1. Whitelist

The following packages have been exported by the DDF Search UI application and are approved for use by third parties:

None.