

1. DDF 2.2.X Home	4
1.1 DDF Users Guide	5
1.1.1 Users Documentation Guide	6
1.1.2 DDF Architecture	7
1.1.3 DDF Catalog	8
1.1.3.1 Catalog Framework	9
1.1.3.2 Catalog Plugins	10
1.1.3.2.1 Pre-Query Plugin	11
1.1.3.2.2 Post-Query Plugin	11
1.1.3.2.3 Pre-Ingest Plugin	11
1.1.3.2.4 Post-Ingest Plugin	12
1.1.3.2.5 Pre-Get Resource Plugin	12
1.1.3.2.6 Post-Get Resource Plugin	12
1.1.3.2.7 Pre-Subscription Plugin	13
1.1.3.2.8 Pre-Delivery Plugin	13
1.1.3.3 Data Components	13
1.1.3.4 Endpoints	16
1.1.3.5 Eventing	16
1.1.3.5.1 Subscription	17
1.1.3.5.2 Delivery Method	17
1.1.3.5.3 Event Processor	17
1.1.3.6 Federation	18
1.1.3.6.1 Federation Strategy	18
1.1.3.7 Operations	18
1.1.3.8 Resource Components	19
1.1.3.8.1 Resource Reader	19
1.1.3.8.2 Resource Writer	19
1.1.3.9 Sources	20
1.1.3.9.1 Catalog Provider	20
1.1.3.9.2 Remote Sources	20
1.1.3.10 Transformers	21
1.1.3.10.1 Input Transformer	21
1.1.3.10.2 Metacard Transformer	22
1.1.3.10.3 Query Response Transformer	22
1.1.4 DDF Content Framework	22
1.1.4.1 Content Framework Architecture	23
1.1.4.2 Content Component Types	23
1.1.4.2.1 Content Data Components	23
1.1.4.2.2 Content Endpoints	24
1.1.4.2.3 Content Framework	24
1.1.4.2.4 Content Operations	24
1.1.4.2.5 Content Plugins	25
1.1.4.2.6 Storage Providers	25
1.1.4.3 Content Framework Execution Flow: Create Content and Catalog Entry from GeoJson File	25
1.1.5 DDF Applications	25
1.1.5.1 DDF Catalog Applications	25
1.1.5.1.1 DDF Catalog Core	26
1.1.5.1.2 Catalog Fanout Framework App	31
1.1.5.1.3 DDF Catalog REST	33
1.1.5.1.4 DDF Catalog OpenSearch	36
1.1.5.1.5 DDF Catalog Solr	41
1.1.5.1.6 DDF Catalog KML	48
1.1.5.1.7 DDF Catalog Schematron	56
1.1.5.1.8 Catalog Transformers	57
1.1.5.2 DDF Content Applications	77
1.1.5.2.1 DDF Content Core	77
1.1.5.2.2 DDF Content REST CRUD Endpoint	81
1.1.5.3 DDF Metrics Applications	87
1.1.5.3.1 Metrics Collecting Application	87
1.1.5.3.2 Metrics Reporting Application	89
1.1.5.4 DDF Mime Applications	96
1.1.5.4.1 DDF Mime Core	96
1.1.5.4.2 DDF Mime Tika	99
1.1.5.5 DDF Platform Application	99
1.1.5.6 DDF Security Applications	100
1.1.5.6.1 Security CAS	100
1.1.5.6.2 Security Core	103
1.1.5.6.3 Security Encryption	105
1.1.5.6.4 Security PDP	106
1.1.5.6.5 Security PEP	108
1.1.5.6.6 Security STS	109
1.1.5.6.7 Security LDAP	113

1.1.6 Web Service Security	116
1.1.6.1 Auditing	118
1.1.6.2 CAS SSO Configuration	125
1.1.6.2.1 Configuring CAS for LDAP	128
1.1.6.2.2 Configuring CAS for X509 User Certificates	131
1.1.6.3 Certificate Management	135
1.1.6.3.1 Cert Config Management	135
1.1.6.3.2 Cert File Management	140
1.1.6.4 Encryption Service	142
1.1.6.5 Redaction and Filtering	142
1.1.6.6 Security Token Service	145
1.1.6.6.1 BinarySecurityToken (CAS) Sample Request/Response	154
1.1.6.6.2 UsernameToken Sample Request/Response	162
1.1.6.6.3 X.509 Token Sample Request/Response	169
1.1.6.7 XACML Policy Decision Point (PDP)	176
1.1.6.8 Expansion Service	188
1.1.6.9 Configuring DDF with WSS using standalone authentication servers.	190
1.1.7 Installing DDF	193
1.1.8 Starting and Stopping	197
1.1.9 Using Console Commands	200
1.1.9.1 Catalog Commands	201
1.1.9.2 Command Scheduler	203
1.1.9.3 Subscriptions Commands	204
1.1.10 Configuration	210
1.1.10.1 Configuration via the Web Console	210
1.1.10.2 Configuration via the System Console	213
1.1.10.3 Configuration via Configuration (.cfg) Files	215
1.1.10.3.1 How do I use port 80 as a non-root user?	217
1.1.10.3.2 HTTP Port Configuration	219
1.1.10.3.3 SSL Enable Services	220
1.1.10.3.4 Enable HTTP Access Logging	222
1.1.11 Configuring a Java Keystore for Secure Communications	223
1.1.12 Hardening	224
1.1.12.1 Directory Permissions	228
1.1.12.2 Deployment Guidelines	229
1.1.12.3 Security	230
1.1.13 DDF Clustering	230
1.1.13.1 DDF Application and Configuration Clustering	230
1.1.14 DDF Load Balancer	236
1.1.15 DDF Data Migration	238
1.1.16 Users Guide Appendix	239
1.1.16.1 Software Versioning	239
1.1.16.2 FAQ, DDF Use of OGC Filter	239
1.1.16.3 Integrator Scenarios	240
1.1.16.3.1 Configuring DDF as a Fanout Proxy	240
1.1.16.3.2 Reconfiguring DDF with a Different Catalog Provider	241
1.1.16.4 Common Problems and Solutions	241
1.1.16.4.1 Blank Web Console	241
1.1.16.4.2 CXF BusException	242
1.1.16.4.3 Distribution does not start	242
1.1.16.4.4 Exception Starting DDF	243
1.1.16.4.5 DDF Becomes Unresponsive to Incoming Requests	243
1.1.16.5 DDF Directory Contents after Installation	244
1.2 DDF Developer's Guide	244
1.2.1 Developer's Documentation Guide	246
1.2.2 Ensuring Compatibility	246
1.2.3 Development Recommendations	247
1.2.4 Working with OSGi	248
1.2.5 Developing Catalog Components	252
1.2.5.1 Catalog Development Fundamentals	252
1.2.5.1.1 Simple Catalog API Implementations	252
1.2.5.1.2 Use of the Whiteboard Design Pattern	253
1.2.5.1.3 Working with the Catalog Framework	253
1.2.5.1.4 Working with Metacards	254
1.2.5.1.5 Working with Queries	256
1.2.5.1.6 Working with Subscriptions	257
1.2.5.1.7 Working with Filters	262
1.2.5.1.8 Working with Transformers	273
1.2.5.1.9 Working with Resources	275
1.2.5.1.10 Commons-DDF Utilities	277
1.2.5.1.11 Working with Settings	278
1.2.5.2 Developing an Endpoint	279

1.2.5.3 Developing a Catalog Plugin	280
1.2.5.4 Developing a Source	288
1.2.5.4.1 Developing a Filter Delegate	291
1.2.5.5 Developing Transformers	293
1.2.5.5.1 Developing an Input Transformer	293
1.2.5.5.2 Developing a Metacard Transformer	296
1.2.5.5.3 Developing a Query Response Transformer	297
1.2.5.5.4 Developing a XSLT Transformer	298
1.2.5.6 Developing a Resource Reader	304
1.2.5.7 Developing a Resource Writer	306
1.2.5.8 Developing a Registry Client	308
1.2.6 Developing at the Framework Level	308
1.2.6.1 Developing Complementary Frameworks	308
1.2.6.2 Developing Console Commands	308
1.2.7 Developing Action Components	309
1.2.8 Developing for the DDF Marketplace	309
1.2.9 Developing Security Token Service Components	312
1.2.9.1 Developing Token Validators	312
1.2.10 Building	313
1.2.11 DDF Camel Components	314
1.2.11.1 Catalog Framework Camel Component	314
1.3 Javadocs	315
1.4 Quick Start	316
1.5 Release Notes	318
1.5.1 2.2.0.RC2	318
1.5.2 2.2.0.RC3	318
1.5.3 2.2.0.RC4	319

DDF 2.2.X Home

Welcome to the home of the Distributed Data Framework (DDF).

The [Quick Start](#) describes how to get up and running quickly.

Please visit the [DDF Users Guide](#) and the [DDF Developer's Guide](#) for more information.

Building

What you need

- * Install [J2SE 7 SDK](#). The build is also compatible with JDK 6.0 Update 29 (or later).
- * Make sure that your JAVA_HOME environment variable is set to the newly installed JDK location, and that your PATH includes %JAVA_HOME%\bin (windows) or \$JAVA_HOME/bin (*nix).
- * Install [Maven 3.0.3](#) (or later). Make sure that your PATH includes the MVN_HOME/bin directory.

How to build

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

Change to the top level directory of DDF source distribution.

```
mvn install
```

This will compile DDF and run all of the tests in the DDF source distribution. It usually takes some time for maven to download required dependencies in the first build.

The distribution will be available under "distribution/ddf/target" directory.

How to Run

- * Unzip the distribution.
- * Run the executable at <distribution_home>/bin/ddf.bat or <distribution_home>/bin/ddf

Additional Information

The [wiki](#) is the right place to find any documentation about DDF.

Discussions can be found on the [Announcements forum](#), [Users forum](#), and [Developers forum](#).

For a DDF binary distribution, please read the release notes on the wiki for a list of supported and unsupported features.

If you find any issues with DDF, please submit reports with JIRA: <https://tools.codice.org/jira/browse/DDF>











For information on contributing to DDF see: <http://www.codice.org/contributing>.

The Website contains additional information at <http://ddf.codice.org>

Many thanks for using DDF.

-- The Codice DDF Development Team

Recently Updated

-  [UsernameToken Sample Request/Response](#)
Apr 20, 2015 • updated by [Scott Tustison](#) • [view change](#)
-  [Starting and Stopping](#)
Jan 30, 2014 • updated by [Joseph Yennaco](#) • [view change](#)
-  [Catalog Geo-formatter Library](#)
Sep 03, 2013 • updated by [Shaun Morris](#) • [view change](#)
-  [SearchUI](#)
Sep 03, 2013 • updated by [Shaun Morris](#) • [view change](#)
-  [Current Version](#)
Aug 23, 2013 • updated by [Shaun Morris](#) • [view change](#)
-  [DDF 2.2.X Home](#)
Aug 23, 2013 • updated by [Shaun Morris](#) • [view change](#)
-  [Distributed Data Framework 2.2.X](#)
Aug 23, 2013 • updated by [Shaun Morris](#)
-  [DDF22](#)
Aug 23, 2013 • attached by [Shaun Morris](#)
-  [OpenSearch Endpoint](#)
Aug 22, 2013 • updated by [Jeff Vettraino](#) • [view change](#)
-  [Auditing](#)
Aug 19, 2013 • updated by [Matthew Ramey](#) • [view change](#)
-  [Security Token Service](#)
Aug 19, 2013 • updated by [Matthew Ramey](#) • [view change](#)
-  [BinarySecurityToken \(CAS\) Sample Request/Response](#)
Aug 19, 2013 • updated by [Matthew Ramey](#) • [view change](#)
-  [Security CAS Client](#)
Aug 17, 2013 • created by [Jeff Vettraino](#)
-  [Configuring DDF with WSS using standalone authentication servers.](#)
Aug 15, 2013 • updated by [Alexander Lamar](#) • [view change](#)
-  [Configuring a Standalone LDAP Server](#)
Aug 15, 2013 • updated by [Alexander Lamar](#) • [view change](#)

DDF Users Guide

Introduction

Distributed Data Framework (DDF) is an agile and modular integration framework. It is primarily focused on data integration, enabling clients to insert, query and transform information from disparate data sources via the DDF Catalog. A Catalog API allows integrators to insert new capabilities at various stages throughout each operation. DDF is designed with several architectural qualities to benefit integrators:

Standardization

- Building on established Free and Open Source Software (FOSS) and open standards avoids vendor lock-in

Extensibility

- System Integrators can extend capabilities by developing and sharing new features

Flexibility

- System Integrators may deploy only those features required

Simplicity of installation and operation

- Unzip and run
- Configuration via a web console
- Simplicity of Development
 - Build simple Plain Old Java Objects (POJOs) and wire them in via a choice of dependency injection frameworks
 - Make use of widely available documentation and components for DDF's underlying technologies
 - Modular development supports multi-organizational and multi-regional teams

Contents

- [Users Documentation Guide](#) — Establishes conventions used throughout the documentation
- [DDF Architecture](#) — Describes ApplicationName at the infrastructure level, including where ApplicationName fits in an architectural stack
- [DDF Catalog](#) — Describes the DDF Catalog application and available options for extending its capabilities
- [DDF Content Framework](#)
- [DDF Applications](#)
- [Web Service Security](#)
- [Installing DDF](#)
- [Starting and Stopping](#)
- [Using Console Commands](#)
- [Configuration](#)
- [Configuring a Java Keystore for Secure Communications](#)
- [Hardening](#)
- [DDF Clustering](#)
- [DDF Load Balancer](#)
- [DDF Data Migration](#)
- [Users Guide Appendix](#) — Includes supplemental information

Users Documentation Guide

Documentation Updates

The most current Distributed Data Framework (DDF) documentation is available at <https://tools.codice.org/wiki/display/DDF/>.

Questions

Questions about DDF or this documentation should be posted to the ddf-users forum (<https://groups.google.com/d/forum/ddf-users>), ddf-announcements forum (<https://groups.google.com/d/forum/ddf-announcements>), or ddf-developers forum (<https://groups.google.com/d/forum/ddf-developers>) where they will be responded to quickly by a member of the DDF team.

Conventions

The following conventions are used within this documentation:

This is a **Tip**, used to provide helpful information.

This is an **Informational Note**, used to emphasize points, remind users of beneficial information, or indicate minor problems in the outcome of an operation.

This is an **Emphasized Note**, used to inform of important information.

This is a **Warning**, used to alert users about the possibility of an undesirable outcome or condition.

Customizable Values

Many values used in descriptions are customizable and should be changed for specific use cases. These values are denoted by < >, and by [[]] when within XML syntax. When using a real value, the placeholder characters should be omitted.

Code Values

Java objects, lines of code, or file properties are denoted with the Monospace font style. Example: `ddf.catalog.CatalogFramework`

Hyperlinks

Some hyperlinks (e.g., <http://localhost:8181/system/console>) within the documentation assume a locally running installation of DDF. Simply change the hostname if accessing a remote host.

DDF Architecture

Architecture Diagram



Unknown macro: 'plantuml'

As depicted in the architectural diagram above, DDF runs on top of an OSGi Framework, a Java Virtual Machine, 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](#).

DDF as a framework hosts DDF Applications, which themselves 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.

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.

OSGi Core

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

- 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 is available on OSGi at <http://www.osgi.org/>.

Built on Apache Karaf

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

- **Web Administration Console** - useful for configuring bundles, installing/uninstalling features, and viewing services
- **System Console** - provides command line administration of the OSGi container. All functionality in the Web Administration Console can also be done via this command line console.
- **Logging** - provides centralized logging to a single log file (data/logs/ddf.log) utilizing log4j
- **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 info at <http://aries.apache.org/modules/blueprint.htm>
- **Spring DM** - An alternative dependency injection framework. DDF is not dependent on specific dependency injection framework. Blueprint is recommended.

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, like JAX-WS and JAX-RS. (More info 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, used by DDF spatial criteria searches. (More info available at <http://geotools.org/>)
- **Joda Time** - provides an enhanced, easier to use, version of Java date and time classes (More info available at <http://joda-time.sourceforge.net>)

For a full list of dependencies view the Software Version Description Document.

Recommended Hardware

Because of its modular nature, DDF can require many or few 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 on a single machine with 16GB of memory and 8 processor cores.

DDF Catalog

Overview

Role

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.

Design

The Catalog is composed of several components and an API that connects them together. The Catalog API is central to DDF's architectural qualities of extensibility and flexibility. The Catalog API consists of Java interfaces that define Catalog functionality and specify interactions between components. These interfaces provide the ability for components to interact without a dependency on a particular underlying implementation, thus allowing the possibility of alternate implementations that can maintain interoperability and share developed components. As such, new capabilities can be developed independently, in a modular fashion, using the Catalog API interfaces and reused by other DDF installations.

Ensuring Compatibility

The Catalog API will evolve, but great care is taken to retain backwards compatibility with developed components. Compatibility is reflected in version numbers. For more information, see the [Software Versioning](#) section in the [Integrator's Guide Appendix](#).

Catalog Architecture

Unable to render {include} The included page could not be found.

Catalog Components

- [Catalog Framework](#) — the core of the Catalog application, routes requests and responses between all Catalog Components
- [Catalog Plugins](#) — process Catalog operations, generally before and after they are executed
- [Data Components](#) — representations of data in the Catalog, primarily metadata represented as a Metacard
- [Endpoints](#) — components that accept external requests and interface with internal components, normalizing the request and denormalizing the response
- [Eventing](#) — allows endpoints (and thus external users) to create a "standing query" and be notified when a matching Metacard is created, updated, or deleted
- [Federation](#) — provides the capability to extend the DDF enterprise to include Remote Sources, which can include other instances of DDF
- [Operations](#) — represent all transactions that occur in the Catalog, including requests and responses
- [Resource Components](#) — used to work with Resources, i.e., the data represented by the cataloged metadata
- [Sources](#) — connect Catalog components to data sources, both local and remote
- [Transformers](#) — transform data to and from various formats

Catalog Framework



Unknown macro: 'plantuml'

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](#).

Example Catalog Frameworks

The Catalog comes with the following Catalog Frameworks out of the box:

[Catalog Framework](#)

[Catalog Fanout Framework](#)

Catalog Framework Sequence Diagrams

Because the Catalog Framework plays a central role to Catalog functionality, it interacts with many different Catalog components. To illustrate these relationships, high level sequence diagrams with notional class names are provided below. These examples are for illustrative purposes only and do not necessarily represent every step in each procedure.

Ingest

The Ingest Service Endpoint, the Catalog Framework, and the [Catalog Provider](#) are key components of the Reference Implementation. The Endpoint bundle implements a Web service that allows clients to create, update, and delete [metacards](#). The Endpoint calls the `CatalogFramework` to execute the operations of its specification. The `CatalogFramework` routes the request through optional PreIngest and PostIngest [Catalog Plugins](#) which may modify the ingest request/response before/after the [Catalog Provider](#) executes the ingest request and provides the response. Note that a `CatalogProvider` **must** be present for any ingest requests to be successfully processed, otherwise a fault is returned.

More details of this flow are shown in the diagram below:



Unknown macro: 'plantuml'

This flow is similar for updating catalog entries, with update requests calling the `update(UpdateRequest)` methods on the Endpoint, Catalog Framework, and [Catalog Provider](#). Similarly for deletion of catalog entries, the delete requests call the `delete(DeleteRequest)` methods on the Endpoint, CatalogFramework, and [Catalog Provider](#).


Error Handling

Any ingest attempts that fail inside the Catalog Framework (whether the failure comes from the Catalog Framework itself, pre-ingest plugin failures, or issues with the Catalog Provider) will be logged to a separate log file for ease of error handling. The file is located at `data/log/ingest_error.log`, and will log the Metacards that fail, their ID and Title name, and the stack trace associated with their failure. By default, successful ingest attempts are not logged, however that functionality can be achieved by setting the log level of the `ingestLogger` to `DEBUG` (note that enabling `DEBUG` can cause a non-trivial performance hit).


To turn off the logging of failed ingest attempts into a separate file execute the following via the command line console: `log:set ERROR ingestLogger`

Query

The Query Service Endpoint, the Catalog Framework, and the [CatalogProvider](#) are key components for processing a query request as well. The Endpoint bundle contains a Web service that exposes the interface to query for Metacards. The Endpoint calls the `CatalogFramework` to execute the operations of its specification. The `CatalogFramework` relies on the `CatalogProvider` to execute the actual query. Optional PreQuery and PostQuery [Catalog Plugins](#) may be invoked by the `CatalogFramework` to modify the query request/response prior to the `Catalog Provider` processing the query request and provides the query response. If a `CatalogProvider` is not configured and no other remote `Source`s are configured a fault will be returned. It is possible to have only remote `Sources` configured and no local `CatalogProvider` configured and be able to execute queries to specific remote `Sources` by specifying the site name(s) in the query request.

 Unknown macro: 'plantuml'

Catalog Plugins

 Unknown macro: 'plantuml'

Description


The [Catalog Framework](#) calls Catalog Plugins to process requests and responses as they enter and leave the Framework.

Types of Catalog Plugins:

- [Pre-Query Plugin](#) — invoked before a query operation is sent to a Source
- [Post-Query Plugin](#) — invoked after a query has been executed successfully, but before the response is returned
- [Pre-Ingest Plugin](#) — invoked before an ingest operation is sent to a Source
- [Post-Ingest Plugin](#) — invoked after data has been created, updated, or deleted
- [Pre-Get Resource Plugin](#) — invoked before a request to retrieve a resource is sent to a Source
- [Post-Get Resource Plugin](#) — invoked after a resource has been retrieved, but before it is returned
- [Pre-Subscription Plugin](#) — invoked before a Subscription is activated by an Event Processor
- [Pre-Delivery Plugin](#) — invoked before a Delivery Method is invoked on a Subscription

Communication Diagrams

Ingest Plugin Flow

 Unknown macro: 'plantuml'

QueryPlugin Flow



Unknown macro: 'plantuml'

Pre-Query Plugin

Usage

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. That is, 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

Usage

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. 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 response is returned to the requesting endpoint.

Pre-Ingest Plugin

Description

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. That is, 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.

Post-Ingest Plugin

Usage

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 can not 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.

Pre-Get Resource Plugin

Usage

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

Usage

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. 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 response is returned to the requesting endpoint.

Pre-Subscription Plugin

Usage

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

Usage

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. That is, 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](#).

Data Components



Unknown macro: 'plantuml'

Metacard

A single instance of metadata in the Catalog (an instance of a Metacard Type). 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.

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

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 for accessing the metadata content type of a Metacard.
CONTENT_TYPE_VERSION	metadata-content-type-version	STRING	Attribute name for accessing the version of the metadata content 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.
RESOURCE_SIZE	resource-size	STRING	Attribute name for accessing the size 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 accessing the target namespace of the 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 <code>image/jpeg</code> and be less than 128 kilobytes.
TITLE	title	STRING	Attribute name for accessing the title of the Metacard.

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

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. See [Working with Metacards](#) for more details.

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 Type Registry

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.

As an 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. See [Working with Metacards](#) in the DDF Developer's Guide for more information.

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.
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

Endpoints

 Unknown macro: 'plantuml'

Description

Endpoints act as a proxy between the client and the [Catalog Framework](#). Endpoints expose the client to the [Catalog Framework](#).

Endpoint interface formats/protocols can include a variety of formats, including (but not limited to):

- SOAP Web services
- RESTful services
- JMS
- RMI
- JSON
- OpenSearch

The Endpoint may transform a client request into a compatible Catalog format and then transform the response into a compatible client format. Endpoints may use [Transformers](#) to perform these transformations. This allows an endpoint to interact with [Source\(s\)](#) that have different interfaces. For example, an [OpenSearch Endpoint](#) can send a query to the [Catalog Framework](#) which could then query a [federated source](#) that has no OpenSearch interface.

Endpoints are meant to be the only client-accessible components in the Catalog.


Examples

The following Endpoints are provided with the default Catalog out of the box:

[RESTful CRUD Endpoint](#)

[OpenSearch Endpoint](#)

Eventing

 Unknown macro: 'plantuml'

Description

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.

Components involved in Eventing include:

- [Subscription](#)
- [Delivery Method](#)
- [Event Processor](#)


Eventing Sequence Diagrams

This section discusses the basic subscription creation flow, how an event is processed by the Catalog, and subsequently the broadcast to subscribers. These flows are for illustrative purposes only and do not necessarily represent every step in each procedure.

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, or 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 [De](#)

[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.

 Unknown macro: 'plantuml'

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. The [Event 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 then 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.

 Unknown macro: 'plantuml'

Subscription

Description

Subscriptions represent "standing queries" in the Catalog. Like a Query, Subscriptions are based on the OGC Filter specification.

Subscription Lifecycle

1. Creation
 - a. Subscriptions are created directly with the [Event Processor](#) or declaratively through use of the [Whiteboard Design Pattern](#).
 - b. The [Event Processor](#) will invoke each [Pre-Subscription Plugin](#) and, if the Subscription is not rejected, the Subscription will be activated.
2. Evaluation
 - a. When a Metacard matching the Subscription is created, updated, or deleted in any [Source](#), each [Pre-Delivery Plugin](#) will be invoked.
 - b. 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.

Subscriptions are not persisted by Default

Subscriptions must be explicitly persisted by an Endpoint and are not persisted by default by the Catalog.

Additional Information

- See [Working with Subscriptions](#) in the [Developer's Guide](#) section.
- See [Use of the Whiteboard Design Pattern](#) in the [Developer's Guide](#) section.

Delivery Method

Description

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

Description

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.

Examples

DDF provides the following implementations of an Event Processor out of the box:

- [Standard Event Processor](#)
- [Fanout Event Processor](#)

Federation


 Unknown macro: 'plantuml'

Description

Federation provides the capability to extend the DDF enterprise to include [Remote Sources](#), which can 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](#) 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](#).

Communication Diagram

 Unknown macro: 'plantuml'

Federation Strategy


Description

A Federation Strategy federates a query to all of the [Remote Sources](#) in the query's list, processes the results in a unique way, and then returns the results to the client. For example, implementations can choose to block until all results return then do a mass sort, or to 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.

Operations

 Unknown macro: 'plantuml'

Description

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, and 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. Refer to the [Developer's Guide](#) for details on using this extensibility.

Resource Components



Unknown macro: 'plantuml'

Description

Resource Components are used to work with Resources, i.e., the data 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

Types of Resource Components include:

- [Resource Reader](#) — retrieves resources associated with Metacards via URIs
- [Resource Writer](#) — stores a resource and produces a URI that can be used to retrieve the resource later

Resource Reader

Description

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 should be exported to the OSGi Service Registry as a `ddf.catalog.resource.ResourceReader`.

Examples

Catalog includes the following Resource Readers out of the box:

- [URL Resource Reader](#)

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>).
- See [Working with Resources](#) and [Developing a Resource Reader](#) in the [Developing Catalog Components](#) section.

Resource Writer

Description

A Resource Writer stores a resource and produces a URI that can be used to retrieve the resource later. The resource URI uniquely locates and identifies the resource.

Resource Writers know how to interact with an underlying data store and store the resource in its proper place. Each implementation can do this

differently, thus 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.

Additional Information

- See [Working with Resources](#) and [Developing a Resource Writer](#) in the [Developing Catalog Components](#) section of the Developer's Guide.

Sources



Unknown macro: 'plantuml'

Description

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

Types of Sources include

- [Catalog Provider](#) — writable data stores that support query, create, update, and delete operations to manage data
- [Remote Sources](#) — read-only data Sources

Catalog Provider

- [Description](#)
- [Usage](#)
- [Additional Information](#)

Description

A Catalog Provider provides an implementation of a searchable and writable catalog. All [Sources](#) support queries, including [Federated Source](#) and [Connected Source](#), 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.

Usage

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) wins.

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.

Additional Information

- [Developing a Source](#)
- [Use of the Whiteboard Design Pattern](#)

Remote Sources

Description

Remote Sources are read-only data [Sources](#) that support query operations but cannot be used to create, update, or delete Metacards.


Types of Remote Sources include:

- [Federated Source](#) — a Remote Source that can be included in federated queries by request or as part of an enterprise query
- [Connected Source](#) — a Source that is included in all local and federated queries, but is hidden from external clients

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.


Federated Source

 Unknown macro: 'plantuml'


Description

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 only support query and site information operations. 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 for event notification when Metacards are created/updated/deleted.

DDF Catalog instances can also be federated to each other, hence a DDF Catalog can also act as a Federated Source to another DDF Catalog.

 Unknown macro: 'plantuml'


Connected Source

 Unknown macro: 'plantuml'

Description

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.


Transformers

 Unknown macro: 'plantuml'

Transformers transform data to and from various formats.

The current Transformers do not support Non-Western Characters (e.g. Hebrew). If the data being transformed contains these characters, they may not be displayed properly. For example, the characters after transformation are not displayed properly (e.g. the word will show up as squares). In other words, Transformers only work for UTF-8. It is recommend not to use international character sets.

Communication Diagram

 Unknown macro: 'plantuml'

The Catalog includes the following types of Transformers:

- [Input Transformer](#) — transforms raw data (text/binary) into a Metacard
- [Metacard Transformer](#) — transforms a Metacard into a text/binary format
- [Query Response Transformer](#) — transforms a query response containing multiple Metacards into a text/binary format

Input Transformer

Description

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 following interface `ddf.catalog.transform.InputTransformer` in order to notify some Catalog components of any new transformers.

Examples

DDF includes the following Input Transformers:

- [Tika Input Transformer](#) — transforms a Microsoft Office, PDF, and OpenOffice documents into a Catalog Metacard
- [GeoJSON Input Transformer](#) — transforms GeoJSON into a Catalog Metacard

Additional Information

- See [Working with Transformers](#) and [Developing an Input Transformer](#) in the [Developing Catalog Components](#) section

Metacard Transformer

Description

A Metacard Transformer transforms a Metacard into a text or binary format.

Examples

DDF includes the following Metacard Transformers:

- [HTML Metacard Transformer](#) — transforms a Metacard into an HTML formatted document
- [XML Metacard Transformer](#) — transforms a Metacard into an XML formatted document
- [GeoJSON Metacard Transformer](#) — transforms a Metacard into GeoJSON text
- [Thumbnail Metacard Transformer](#) — retrieves the thumbnail bytes of a Metacard
- [Metadata Metacard Transformer](#) — returns the Metacard.METADATA attribute value when given a Metacard.
- [Resource Metacard Transformer](#) — retrieves the resource bytes of a Metacard product

Additional Information

- See [Working with Transformers](#) in the [Developing Catalog Components](#) section

Query Response Transformer

Description

A Query Response Transformer transforms a query response containing multiple Metacards into a text or binary format.

Examples

DDF includes the following Query Response Transformers:

- [Atom Query Response Transformer](#) — transforms a Query Response into an Atom 1.0 <http://tools.ietf.org/html/rfc4287> feed
- [HTML Query Response Transformer](#) — transforms a Query Response into an HTML formatted document
- [XML Query Response Transformer](#) — transforms a Query Response into an XML formatted document
- [SearchUI](#) — a `QueryResponseTransformer` that not only provides results in a html format but also provides a convenient, simple querying user interface

Additional Information

- See [Working with Transformers](#) and [Developing a Query Response Transformer](#) in the [Developing Catalog Components](#) section

DDF Content Framework

Overview

The DDF Content Framework is a framework for storing, reading, processing, and transforming content information. Content information is defined as files that the client wants:

- parsed and a Metacard created that is subsequently used to create a catalog entry in the Metadata Catalog
- stored in the DDF Content Repository

The files passed into the DDF [Content Framework](#) can be of any type, e.g., NITF, PDF, Microsoft Word, etc., as long as their mime type can be resolved and an [Input Transformer](#) exists to parse their content into a Metacard and the generated Metacard satisfies the constraints of the

catalog provider that the generated Metacard will be inserted into. For example, if the [Tika Input Transformer](#) is installed, then Microsoft Office documents and PDF files can be transformed into metacards. If the [Solr catalog provider](#) is being used, then the generated Metacard can be successfully inserted.

Clients typically perform create, read, update, and delete (CRUD) operations against the content repository. At the core of the Content functionality is the Content Framework, which routes all requests and responses through the system, invoking additional processing per the system configuration.

The DDF Content Framework is composed of several components and an API that connects them together. The Content API consists of the Java interfaces that define DDF functionality. These interfaces provide the ability for components to interact without a dependency on a particular underlying implementation, thus allowing the possibility of alternate implementations that can maintain interoperability and share developed components. As such, new capabilities can be developed independently, in a modular fashion, using the Content API interfaces and reused by other DDF installations.

The DDF Content API will evolve with DDF itself, but great care is taken to retain backwards compatibility with developed components. Compatibility is reflected in version numbers. For more information, see the [Software Versioning](#) section in the [Appendix](#).

Content Framework Architecture

Architecture



Unknown macro: 'plantuml'

Design

The DDF Content Framework design consists of several major components, namely Endpoints, Input Transformers, the core Content Framework, Storage Providers, and Content Plugins.

The Endpoints provide external clients access to the Content Framework. Input Transformers convert incoming content into a Metacard. The core Content Framework routes requests and responses through the system. The Storage Providers provide storage of the content to specific types of storage, e.g., file system, relational database, XML database, etc. The Content Plugins provide pluggable functionality that can be executed after the content has been stored/update/deleted but before the response has been returned to the client.

The UML class diagrams below illustrates how these components are related to each other.



Unknown macro: 'plantuml'

DDF Content API Packaging

The Content API consists of the Java interfaces that define the methods that this API supports. These interfaces and how they are packaged is illustrated below.



Unknown macro: 'plantuml'

DDF Content API Reference Implementation

The Content API Reference Implementation consists of the Java classes that implement the methods defined in the Content API. These classes and how they are packaged is illustrated below.



Unknown macro: 'plantuml'

Content Component Types

- [Content Data Components](#) — domain objects representing the content to be stored
- [Content Endpoints](#) — components that accept external requests and interface with the internal components, storing content in the content repository and/or creating catalog entries
- [Content Framework](#) — the core of the Content application, routes requests and responses between the Content Components
- [Content Operations](#) — represents all transactions that occur in the Content Framework, specifically requests and responses
- [Content Plugins](#) — process Content operations after content storage, generally to parse content's metadata and create a Metacard from it to insert into the catalog provider
- [Storage Providers](#) — proxies to storage mechanisms used to store the content

Content Data Components



Unknown macro: 'plantuml'

Content Item

Content Item is the domain object, populated by the [Content Endpoint](#) from the client request, that represents the information about the content to be stored in the [Storage Provider](#). A Content Item encapsulates the content's globally unique ID, mime type, and input stream (i.e., the actual content).

Content Endpoints



Unknown macro: 'plantuml'

Description

Content Endpoints act as a proxy between the client and the [Content Framework](#). Endpoints expose the client to the [Content Framework](#).

Endpoint interface formats/protocols can include a variety of formats, including (but not limited to):

- SOAP Web services
- RESTful services
- JMS
- RMI
- JSON
- OpenSearch

Content Endpoints provide the capability to create, read, update, and delete content in the content repository, as well as create, update, and delete Metacards corresponding to the content in the Metadata Catalog.

Endpoints are the only client-accessible components in DDF.

Examples

The following Endpoints are provided with the Content Framework out of the box:

[Content REST CRUD Endpoint](#)

Content Framework

Description

The Content Framework wires all Content Components together via OSGi and the Content API. It handles all Content Operations requested by [Endpoints](#), invoking [Content Plugins](#) as needed, and for most [Operations](#), sending the Request to a [Storage Provider](#) for execution.



Unknown macro: 'plantuml'

Examples

The DDF Content comes with the following Content Frameworks out of the box:

[Standard Content Framework](#)

Content Operations



Unknown macro: 'plantuml'

Description


The DDF Content provides the capability to read, create, update, and delete content from the DDF Content Repository.

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 [Content Framework](#)'s operation to communicate with the [Storage Providers](#) and [Content Plugins](#). The response is the output from the execution of the Operation that is returned to the client, and contains all of the data returned by the [Storage Providers](#) and [Content Plugins](#). For each Operation there is an associated request/response pair, e.g., the CreateRequest and CreateResponse pair for the [Content Framework](#)'s create 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 Content API, helping to maintain backwards compatibility. Refer to the [Developer's Guide](#) for details on using this extensibility.

Content Plugins

 Unknown macro: 'plantuml'


Description

The [Content Framework](#) calls Content Plugins to process requests after they have been processed by the [Storage Provider](#). If the request does not specify content storage, only processing, then the Content Plugins are called immediately by the [Content Framework](#).

Types of Content Plugins available out-of-the-box:

[Content Cataloger Plugin](#)

Storage Providers

 Unknown macro: 'plantuml'

Description

Storage Providers act as a proxy between the [Content Framework](#) and the mechanism storing the content, e.g., file system, relational database, etc.. Storage Providers expose the storage mechanism to the [Content Framework](#).

Storage Providers provide the capability to the [Content Framework](#) to create, read, update, and delete content in the content repository


Examples

The following Storage Providers are provided with the Content Framework out of the box:

[File System Storage Provider](#)

Content Framework Execution Flow: Create Content and Catalog Entry from GeoJson File

The UML sequence diagram below illustrates how the DDF Content components interact when storing a GeoJson file and creating a catalog entry for it by parsing the GeoJson file and creating a Metacard.

 Unknown macro: 'plantuml'

DDF Applications

- [DDF Catalog Applications](#)
- [DDF Content Applications](#)
- [DDF Metrics Applications](#)
- [DDF Mime Applications](#)
- [DDF Platform Application](#)
- [DDF Security Applications](#)

DDF Catalog Applications

- [DDF Catalog Core](#)
 - [Catalog API and Global Settings](#) — The Content API is an OSGi bundle that contains the Java classes and interfaces that allow the various Content Component Types to integrate with each other.
 - [Standard Catalog Framework](#) — provides the reference implementation of a Catalog Framework
 - [Standard Event Processor](#) — creates and deletes subscriptions, applies the criteria for each subscription to ingest events, and sends events to registered consumers when the criteria is matched
 - [URL Resource Reader](#) — obtains a resource given an http, https, or file-based URL
 - [Sorted Federation Strategy](#) — the default Federation Strategy which returns results sorted by the sorting parameter specified in the federated query
 - [Fanout Event Processor](#) — Event Processor for fanout configuration
 - [Metacard Groomer](#) — The Metacard Groomer Pre-Ingest Plugin makes modifications to CreateRequest and UpdateRequest metacards.
- [Catalog Fanout Framework App](#) — provides an implementation of the Catalog Framework that acts as a proxy, federating requests to all available sources
- [DDF Catalog REST](#)

- **RESTful CRUD Endpoint** — allows clients to perform CRUD operations on DDF using REST, a simple architectural style that performs communication using HTTP.
- **DDF Catalog OpenSearch**
 - **OpenSearch Endpoint** — provides a RESTful endpoint that a client accesses to send OpenSearch formatted queries
 - **OpenSearch Source** — allows the DDF Catalog to federate queries via OpenSearch to a CDR-compliant Search Service
- **DDF Catalog Solr**
 - **Solr Catalog Provider Apps** — implementation of a CatalogProvider using Apache Solr as the data store.
 - **Standalone Solr Server App** — an Apache Solr instance as a Catalog data store within the distribution
 - **Solr Standalone Server Backup**
- **DDF Catalog KML**
 - **KML Query Response Transformer** — transforms a Query Response into a KML formatted document
 - **KML Metacard Transformer** — transforms a Metacard into a KML formatted document
 - **KML Network Link Endpoint** — allows a user to generate a View-based KML Query Results Network Link that can be opened with Google Earth, establishing a dynamic connection between Google Earth and DDF
- **DDF Catalog Schematron**
 - **Schematron Pre-Ingest Plugin Framework** — provides a pre-ingest interceptor that validates the incoming request against a Schematron ruleset (or rule sets)
- **Catalog Transformers**
 - **XSLT Transformer** — allows developers to create light-weight Query Response Transformers
<https://wiki.macefusion.com/display/DDF/Query+Response+Transformer> and Metacard Transformers
<https://wiki.macefusion.com/display/DDF/Metacard+Transformer> using only a bundle header and XSLT files.
 - **Included Input Transformers** — convert input data into Catalog Metacards
 - **Tika Input Transformer** — transforms a Microsoft Office, PDF, and OpenOffice documents into a Catalog Metacard
 - **GeoJSON Input Transformer** — transforms GeoJSON into a Catalog Metacard
 - **Included Metacard Transformers** — convert Metacards into other data formats
 - **HTML Metacard Transformer** — transforms a Metacard into an HTML formatted document
 - **XML Metacard Transformer** — transforms a Metacard into an XML formatted document
 - **GeoJSON Metacard Transformer** — transforms a Metacard into GeoJSON text
 - **Thumbnail Metacard Transformer** — retrieves the thumbnail bytes of a Metacard
 - **Metadata Metacard Transformer** — returns the Metacard.METADATA attribute value when given a Metacard.
 - **Resource Metacard Transformer** — retrieves the resource bytes of a Metacard product
 - **Included Query Response Transformers** — convert Query Responses into other data formats
 - **Atom Query Response Transformer** — transforms a Query Response into an Atom 1.0
<http://tools.ietf.org/html/rfc4287> feed
 - **HTML Query Response Transformer** — transforms a Query Response into an HTML formatted document
 - **XML Query Response Transformer** — transforms a Query Response into an XML formatted document
 - **SearchUI** — a QueryResponseTransformer that not only provides results in a html format but also provides a convenient, simple querying user interface
 - **Catalog Geo-formatter Library** — Geo library to help with conversion of geometry objects into various formats such as GeoJson, GeoRSS, etc.

DDF Catalog Core

- **Catalog API and Global Settings** — The Content API is an OSGi bundle that contains the Java classes and interfaces that allow the various Content Component Types to integrate with each other.
- **Standard Catalog Framework** — provides the reference implementation of a Catalog Framework
- **Standard Event Processor** — creates and deletes subscriptions, applies the criteria for each subscription to ingest events, and sends events to registered consumers when the criteria is matched
- **URL Resource Reader** — obtains a resource given an http, https, or file-based URL
- **Sorted Federation Strategy** — the default Federation Strategy which returns results sorted by the sorting parameter specified in the federated query
- **Fanout Event Processor** — Event Processor for fanout configuration
- **Metacard Groomer** — The Metacard Groomer Pre-Ingest Plugin makes modifications to CreateRequest and UpdateRequest metacards.

Catalog API and Global Settings

Description

The Content API is an OSGi bundle that contains the Java classes and interfaces that allow the various **Content Component Types** to integrate with each other.

These two bundles, along with several other lower level bundles, are packaged as the `content-core` feature.

Additionally, DDF has several settings that are used system-wide, regardless of the components that are installed.

- [Configuration](#)
 - [Configurable Properties](#)

Configuration

Configuration can be performed using the processes described in the [Configuration](#) section. The configurable properties for the catalog-wide configuration are accessed from Configuration -> *Platform Global Configuration* in the Admin Console.

Configurable Properties

Title	Property	Type	Description	Default Value	Required
Protocol	protocol	String	Default protocol that should be used to connect to this machine.	http	yes
Host	host	String	The host name or IP address of the machine that DDF is running on. Do not enter localhost.		yes
Port	port	String	The port that DDF is running on.		yes
Trust Store	trustStore	String	The trust store used for outgoing SSL connections. Path is relative to karaf.home.	etc/keystores/clientTruststore.jks	no
Trust Store Password	trustStorePassword	String	The password associated with the trust store.	changeit (encrypted)	no
Key Store	keyStore	String	The key store used for outgoing SSL connections. Path is relative to karaf.home.	etc/keystores/clientKeystore.jks	no
Key Store Password	keyStorePassword	String	The password associated with the key store.	changeit (encrypted)	no
Site Name	id	String	The site name for this DDF instance.	ddf.distribution	yes
Version	version	String	The version of DDF that is running. This value should not be changed from the factory default.	DDF 2.2.1	yes
Organization	organization	String	The organization responsible for this installation of DDF	Codice Foundation	yes

Standard Catalog Framework

- [Description](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
 - [Configurable Properties](#)
- [Implementation Details](#)
 - [Exported Services](#)
 - [Imported Services](#)
- [Known Issues](#)

Description

The Standard Catalog Framework provides the reference implementation of a [Catalog Framework](#) that implements all requirements of the DDF Catalog API. `CatalogFrameworkImpl` is the implementation of the DDF Standard Catalog Framework.

Usage

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, then the site information returned includes site info for the catalog framework with no content types included.

Installation and Uninstallation

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.

Configuration

DDF

Configurable Properties

Catalog Standard Framework

Property	Type	Description	Default Value	Required
poolSize	Integer	The federation thread pool size (0 for unlimited)	0	yes

Managed Service PID	ddf.catalog.CatalogFrameworkImpl
Managed Service Factory PID	N/A

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.util.DdfConfigurationWatcher		
ddf.catalog.event.EventProcessor		
ddf.catalog.plugin.PostIngestPlugin		

Imported Services

Registered Interface	Availability	Multiple
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.PreIngestPlugin	optional	true
ddf.catalog.plugin.PreQueryPlugin	optional	true
ddf.catalog.plugin.PreResourcePlugin	optional	true

<code>ddf.catalog.plugin.PreSubscriptionPlugin</code>	optional	true
<code>ddf.catalog.resource.ResourceReader</code>	optional	true
<code>ddf.catalog.source.ConnectedSource</code> ,	optional	true
<code>ddf.catalog.source.FederatedSource</code>	optional	true
<code>org.osgi.service.event.EventAdmin</code>		false

Known Issues

None

Standard Event Processor

Description

The Standard Event Processor is an implementation of the [Event Processor](#) and provides the ability to create/delete [subscriptions](#). As events are generated by the DDF [Catalog Framework](#) as metacards are created/updated/deleted 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

Installation and Uninstallation

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.

URL Resource Reader

- [Description](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Implementation Details](#)
- [Known Issues](#)

Description

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`.

Usage

`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.

Installation and Uninstallation

`URLResourceReader` is installed by default with the DDF Catalog.

Configuration

This `URLResourceReader` has no configurable properties. It can only be installed or uninstalled.

Implementation Details

Supported Schemes	<ul style="list-style-type: none">• <code>http</code>• <code>https</code>• <code>file</code>
-------------------	--

If a file-based URL is passed to the `URLResourceReader`, that file path needs to be accessible by the DDF instance.

Known Issues

None

Sorted Federation Strategy

Description

The Sorted 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 or Descending.

The default sorting value/order automatically used is Relevance Descending.

The `SortedFederationStrategy` expects the results returned from the [Sources](#) 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 `SortedFederationStrategy` expects that result to come at the end of the result list.

Configuration

The Sorted Federation Strategy configuration can be found in the webconsole under Configuration -> Catalog Sorted Federation Strategy.

Property	Type	Description	Default Value	Required
<code>maxStartIndex</code>	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: <ul style="list-style-type: none">• How much memory is allocated to the DDF Server• How many sites are being federated with.	50000	yes

Managed Service PID	<code>ddf.catalog.SortedFederationStrategy</code>
Managed Service Factory PID	N/A

Fanout Event Processor

Description

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](#).

Installation and Uninstallation

The Fanout Event Processor is automatically installed/uninstalled when the [Catalog Fanout Framework App](#) is installed/uninstalled.

Known Issues

None

Metacard Groomer

- [Description](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Known Issues](#)

Description

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 timestamp
- Overwrites the Metacard.MODIFIED date with a current timestamp

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.CREATEDdate is not null.
- Overwrites the Metacard.MODIFIED date with a current timestamp

Usage

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

Installation and Uninstallation

This plugin can be installed and uninstalled using the normal processes described in the [Configuration](#) section.

Configuration

No configuration is necessary for this plugin.

Known Issues

None.

Catalog Fanout Framework App

- [Description](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
 - [Configurable Properties](#)
- [Implementation Details](#)
 - [Exported Services](#)
 - [Imported Services](#)
- [Known Issues](#)

Description

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, hence it does not allow catalog modifications to take place. Therefore create, update, and delete operations are not supported.

 Unknown macro: 'plantuml'

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

Usage

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.

Installation and Uninstallation

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](#).

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.

Configuration

The Fanout Catalog Framework can be configured using the normal processes described in [Configuration](#).

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
Federation Thread Pool Size (0 for unlimited)	poolSize	Integer	The federation thread pool size (0 for unlimited)	0	yes
Default Timeout (in milliseconds)	defaultTimeout	Integer	The maximum amount of time to wait for a response from the Sources	60000	yes
Managed Service PID	ddf.catalog.impl.service.fanout.FanoutCatalogFramework				
Managed Service Factory PID	N/A				

Implementation Details

Exported Services

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

Imported Services

Registered Interface	Availability	Multiple
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.PreIngestPlugin	optional	true
ddf.catalog.plugin.PreQueryPlugin	optional	true
ddf.catalog.plugin.PreResourcePlugin	optional	true
ddf.catalog.plugin.PreSubscriptionPlugin	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

Known Issues

None

DDF Catalog REST

- [RESTful CRUD Endpoint](#) — allows clients to perform CRUD operations on DDF using REST, a simple architectural style that performs communication using HTTP.

RESTful CRUD Endpoint

- [Description](#)
- [Using the REST CRUD Endpoint](#)
 - [Metacard Transforms with the REST CRUD Endpoint](#)
 - [Metacard Transforms Available in DDF](#)
 - [InputTransformers](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Implementation Details](#)
 - [Imported Services](#)
 - [Exported Services](#)
- [Known Issues](#)

Description

The Catalog REST Endpoint allows clients to perform CRUD operations on the Catalog using REST, a simple architectural style that performs communication using HTTP. The URL exposing the REST functionality is located at `http://<HOST>:<PORT>/services/catalog`, where `HOST` is the IP address of where the distribution is installed and `PORT` is the port number on which the distribution is listening.

Using the REST CRUD Endpoint

The RESTful CRUD Endpoint provides the capability to query, create, update, and delete metacards in the catalog provider as follows:

Operation	HTTP Request	Details	Example URL
create	HTTP POST	HTTP request body contains the input to be ingested. See InputTransformers for more information.	<code>http://<DISTRIBUTION_HOST>:<DISTRIBUTION_PORT>/services/catalog</code>
update	HTTP PUT	The ID of the Metacard to be updated is appended to the end of the URL. The updated metadata is contained in the HTTP body.	<code>http://<DISTRIBUTION_HOST>:<DISTRIBUTION_PORT>/services/catalog/<metacardId></code> where <code><metacardId></code> is the Metacard.ID of the metacard to be updated
delete	HTTP DELETE	The ID of the Metacard to be deleted is appended to the end of the URL.	<code>http://<DISTRIBUTION_HOST>:<DISTRIBUTION_PORT>/services/catalog/<metacardId></code> where <code><metacardId></code> is the Metacard.ID of the metacard to be deleted
read	HTTP GET	The ID of the Metacard to be retrieved is appended to the end of the URL. By default, the response body will include the XML representation of the Metacard.	<code>http://<DISTRIBUTION_HOST>:<DISTRIBUTION_PORT>/services/catalog/<metacardId></code> where <code><metacardId></code> is the Metacard.ID of the metacard to be retrieved
federated read	HTTP GET	The SOURCE ID of a federated source is appended in the URL before the ID of the Metacard to be retrieved is appended to the end.	<code>http://<DISTRIBUTION_HOST>:<DISTRIBUTION_PORT>/services/catalog/sources/<sourceId>/<metacardId></code> where <code><sourceId></code> is the FEDERATED SOURCE ID and <code><metacardId></code> is the Metacard.ID of the Metacard to be retrieved

Note that for all RESTful CRUD commands only one metacard ID is supported in the URL, i.e., bulk operations are not supported.

Interacting with the REST CRUD Endpoint

Any web browser can be used to perform a REST read. Various other tools and libraries can be used to perform the other HTTP operations on the REST endpoint (e.g., `soapUI`, `cURL`, etc.)

Metacard Transforms with the REST CRUD Endpoint

The `read` operation can be used to retrieve metadata in different formats.

1. Install the appropriate feature for the desired transformer. If desired transformer is already installed such as those that come out of the box (`xml`, `html`, etc), then skip this step.
2. Make a read request to the REST URL specifying the catalog id.
3. Add a transform query parameter to the end of the URL specifying the shortname of the transformer to be used (e.g., `transform=kml`).

Example:

```
http://<DISTRIBUTION_HOST>:<DISTRIBUTION_PORT>/services/catalog/<meta  
cardId>?transform=<TRANSFORMER_ID>
```

Transforms also work on read operations for metacards in federated sources.

```
http://<DISTRIBUTION_HOST>:<DISTRIBUTION_PORT>/services/catalog  
/sources/<sourceId>/<metacardId>?transform=<TRANSFORMER_ID>
```

Metacard Transforms Available in DDF

Unable to render {children}. Page not found: DDF:Included Metacard Transformers.

MetacardTransformers can be added to the system at any time. This endpoint can make use of any registered MetacardTransformers.

InputTransformers

This REST Endpoint uses InputTransformers to create metacards from a `create` or a `HTTP POST` operation. The REST Endpoint dynamically finds InputTransformers that support the `Content-Type` stated in the HTTP header of a `HTTP POST`. InputTransformers register as Services with a list of mime-types. The REST Endpoint receives a list of InputTransformers that match the `Content-Type` and one-by-one calls the InputTransformers until a transformer is successful and creates a Metacard. For instance, if GeoJSON was in the body of the `HTTP POST`, then the `HTTP Content-Type` header would need to include `application/json` in order to match the mime-type [GeoJSON Input Transformer](#) supports.

The following are the included InputTransformers:

- [Tika Input Transformer](#) — transforms a Microsoft Office, PDF, and OpenOffice documents into a Catalog Metacard
- [GeoJSON Input Transformer](#) — transforms GeoJSON into a Catalog Metacard

InputTransformers can be added to the system at any time.

Installation and Uninstallation

The RESTful CRUD Endpoint can be installed and uninstalled using the normal processes described in the [Configuration](#) section.

Configuration

The RESTful CRUD Endpoint has no configurable properties. It can only be installed or uninstalled.

Implementation Details

Imported Services

Registered Interface	Availability	Multiple
<code>ddf.mime.MimeTypeToTransformerMapper</code>	required	false
<code>ddf.catalog.CatalogFramework</code>	required	false
<code>ddf.catalog.filter.FilterBuilder</code>	required	false

Exported Services

Registered Interface	Service Property	Value
<code>ddf.action.ActionProvider</code>	<code>id</code>	<code>catalog.data.metacard.view</code>

ddf.catalog.util.DdfConfigurationWatcher		
--	--	--

Known Issues

None

DDF Catalog OpenSearch

- [OpenSearch Endpoint](#) — provides a RESTful endpoint that a client accesses to send OpenSearch formatted queries
- [OpenSearch Source](#) — allows the DDF Catalog to federate queries via OpenSearch to a CDR-compliant Search Service

OpenSearch Endpoint

- [Description](#)
- [Using the OpenSearch Endpoint](#)
 - [Parameter List](#)
 - [Supported Complex Contextual Query Format](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [OpenSearch Description Document](#)

Implementation Details

- [Imported Services](#)
- [Exported Services](#)
- [Example Output](#)
- [Known Issues](#)

Description

The OpenSearch Endpoint provides a DDF endpoint that a client accesses to send query parameters and receive search results.

This endpoint uses the input query parameters to create an OpenSearch query. The client does not need to specify all of the query parameters, only the query parameters of interest.

This endpoint is a JAX-RS RESTful service and is compliant with the CDR IPT BrokeredSearch, CDR IPT OpenSearch, and OpenSearch Specifications. For more information on its parameters view the [OpenSearch Description Document](#) section below

Using the OpenSearch Endpoint

Once installed, the OpenSearch endpoint is accessible from `http://<DDF_HOST>:<DDF_PORT>/services/catalog/query`.

Using the endpoint

From Code:

The OpenSearch specification defines a file format to describe an OpenSearch endpoint. This file is XML-based and is used to programmatically retrieve a site's endpoint, as well as the different parameter options a site holds. The parameters are defined via the OpenSearch and CDR IPT Specifications.

From a Web Browser:

Many modern web browsers currently act as OpenSearch clients. The request call is an HTTP GET with the query options being parameters that are passed.

Example of an OpenSearch request:

```
http://<ddf_host>:8181/services/catalog/query?q=Predator
```

This request performs a full-text search for the phrase 'Predator' on the DDF providers and provides the results as Atom-formatted XML for the web browser to render.

Parameter ListMain OpenSearch Standard

OS Element	HTTP Parameter	Possible Values	Comments
searchTerms	q	URL-encoded string	Complex contextual search string.
count	count	integer >= 0	Maximum # of results to retrieve default: 10

startIndex	start	integer >= 1	Index of first result to return. default: 1 This value uses a one based index for the results.
format	format	requires a transformer shortname as a string, possible values include when available <ul style="list-style-type: none"> atom html kml see Included Query Response Transformers for more possible values	default: atom

Temporal Extension

OS Element	HTTP Parameter	Possible Values	Comments
start	dtstart	RFC-3399 -defined value	yyyy-MM-dd'T'HH:mm:ss.SSSZZ
end	dtend	RFC-3399 -defined value	yyyy-MM-dd'T'HH:mm:ss.SSSZZ

The start and end temporal criteria **must** be of the format specified above. Other formats are currently not supported. Example: 2011-01-01T12:00:00.111-04:00.

The start and end temporal elements are based on modified timestamps for a metacard.

Geospatial Extension

These Geospatial query parameters are used to create a geospatial INTERSECTS query, where INTERSECTS = geometries that are not DISJOINT of the given geospatial parameter.

OS Element	HTTP Parameter	Possible Values	Comments
lat	lat	EPSG:4326 decimal degrees	Expects a latitude and a radius to be specified.
lon	lon	EPSG:4326 decimal degrees	Expects a longitude and a radius to be specified.
radius	radius	Meters along the Earth's surface > 0	Used in conjunction with lat and lon query parameters.
polygon	polygon	clockwise lat lon pairs ending at the first one	example: -80, -170, 0, -170, 80, -170, 80, 170, 0, 170, -80, 170, -80, -170 According to the OpenSearch Geo Specification this is deprecated . Use geometry instead.
box	bbox	4 comma-separated EPSG:4326 decimal degrees	west, south, east, north
geometry	geometry	WKT Geometries: POINT, POLYGON, MULTIPOINT, MULTIPOLYGON	Examples: POINT(10 20) where 10 is the longitude and 20 is the latitude. POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10)). 30 is longitude and 10 is latitude for the first point. Make sure to repeat the starting point as the last point to close the polygon.

Extensions

OS Element	HTTP Parameter	Possible Values	Comments
sort	sort	sbfield: 'date' or 'relevance' sborder: 'asc' or 'desc'	sort=<sbfield>:<sborder> default: relevance:desc Sorting by date will sort the effective date.
maxResults	mr	Integer >= 0	Maximum # of results to return. If count is also specified, the count value will take precedence over the maxResults value
maxTimeout	mt	Integer > 0	Maximum timeout (milliseconds) for query to respond default: 300000 (5 minutes)

Federated Search

OS Element	HTTP Parameter	Possible Values	Comments
------------	----------------	-----------------	----------

routeTo	src	(varies depending on the names of the sites in the federation)	comma delimited list of site names to query. Also can specify <code>src=local</code> to query the local site. If <code>src</code> is not provided, the default behavior is to execute an enterprise search to the entire federation.
---------	-----	--	--

DDF Extensions

OS Element	HTTP Parameter	Possible Values	Comments
dateOffset	dtoffset	integer > 0	Specifies an offset, backwards from the current time, to search on the modified time field for entries. Defined in milliseconds.
type	type	nitf	Specifies the type of data to search for.
version	version	20,30	Comma-delimited list of version values to search for.
selector	selector	//namespace:example,//example	Comma-delimited list of XPath string selectors that narrow down the search.

Supported Complex Contextual Query Format

The contextual query format is based on the "IC/DoD Keyword Query Language Specification, V2.0" DRAFT 9/4/2012.

The OpenSearch Endpoint supports the following operators: AND, OR, and NOT. These operators are case sensitive. Implicit ANDs are also supported.

Using parenthesis to change the order of operations is supported. Using quotes to group keywords into literal expressions is supported.

The following EBNF describes the grammar used for the contextual query format.

OpenSearch Complex Contextual Query EBNF

```

keyword query expression = optional whitespace, term, {boolean operator,
term}, optional whitespace;
boolean operator = or | not | and;
and = (optional whitespace, "AND", optional whitespace) | mandatory
whitespace;
or = (optional whitespace, "OR", optional whitespace);
not = (optional whitespace, "NOT", optional whitespace);
term = group | phrase | keyword;
phrase = optional whitespace, '"', optional whitespace, keyword, { optional
whitespace, keyword}, optional whitespace, '"';
group = optional whitespace, '(', optional whitespace, keyword query
expression, optional whitespace, ')';
optional whitespace = { ' ' };
mandatory whitespace = ' ', optional whitespace;
valid character = ? any printable character ? - (' ' | '(' | ')' | " ");
keyword = valid character, {valid character};

```

Installation and Uninstallation

The OpenSearch Endpoint can be installed and uninstalled using the normal processes described in the [Configuration](#) section.

Configuration

The OpenSearch Endpoint has no configurable properties. It can only be installed or uninstalled.

OpenSearch Description Document

The OpenSearch Description Document is an XML file is found inside of the OpenSearch Endpoint bundle and is named `ddf-os.xml`

Implementation Details

Imported Services

Registered Interface	Availability	Multiple
ddf.catalog.CatalogFramework	required	false
ddf.catalog.filter.FilterBuilder	required	false

Exported Services

Registered Interface	Service Property	Value
ddf.catalog.util.DdfConfigurationWatcher		

Example Output

The default output for OpenSearch is Atom. Detailed documentation on Atom output, including query result mapping and example output, can be found on the [Atom Query Response Transformer](#) page.

Known Issues

OpenSearch Source

- [Description](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
 - [Configuring the OpenSearch Source](#)
- [Source Details](#)
 - [Default Security Settings \(applicable to all OpenSearch Sources\)](#)
- [Query Format](#)
 - [OpenSearch Parameter to DDF Query Mapping](#)

Implementation Details

- [Imported Services](#)
 - [Exported Services](#)
- [Known Issues and Limitations](#)

Description

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. See <http://www.dni.gov/index.php/about/organization/chief-information-officer/cdr-search> for the CDR Specifications. The OpenSearch Source does not provide a [Connected Source](#) interface.

The OpenSearch Source converts a query into OpenSearch format and then sends that request to the CDR-compliant Search Service. It then accepts the response, formatted in Atom, and translates the Atom entries to DDMS Resources (using the [Atom Query Response Transformer](#)). Existing DDMS is used from the Atom entries when available. If it is not available, then a new DDMS document is created from the Atom information. If no (or incomplete) security markings are present, then this bundle adds security markings per the security settings that were configured in the Default Security Settings feature. The OpenSearch Source then splits out the DDMS documents into results, which are sent back to the endpoint/client via the [Catalog Framework](#).

Usage

Use the OpenSearch Source if querying a CDR-compliant Search Service is desired. The OpenSearch Source, which is used for federating to other DDFs.

Installation and Uninstallation

The OpenSearch Source can be installed and uninstalled using the normal processes described in the [Configuration](#) section.

Configuration

This Component can be configured using the normal processes described in the [Configuration](#) section.

The configurable properties for the OpenSearch Source are accessed from the *Catalog OpenSearch Federated Source* Configuration in the Admin Console.

Configuring the OpenSearch Source

Configurable Properties

Title	Property	Type	Description	Default Value
Source Name	shortname	String		DDF-OS
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={search?}
Always perform local query	localQueryOnly	Boolean	Always performs a local query by setting <code>src=local</code> OpenSearch parameter in endpoint URL. This must be set if federating to another DDF.	false
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
Truststore File Location	trustStoreLocation	String	Truststore is only required for secure OpenSearch endpoints. This is the local file location of a java keystore that contains the trusted root certificates for the server being called. For the official servers this is the "store.jks" file attached to the Configuring a Java Keystore for Secure Communications page.	trustStore.jks
Truststore Password	trustStorePassword	String	The password of the above truststore.	password
Keystore File Location	keyStoreLocation	String	This is the local file location of a java keystore that contains the client certificates sent to the server and used for client authentication. This should be a personal ECA Cert that has been exported and converted from PKCS12 to a Java keystore (.jks).	keyStore.jks
Keystore Password	keyStorePassword	String	The password of the above keystore. This should be the same password that used when exporting the client cert.	password

Information on creating keystores and truststores can be found on the [Configuring a Java Keystore for Secure Communications](#) page.

Source Details

Default Security Settings (applicable to all OpenSearch Sources)

These settings are used to provide default security settings for the Title, Description, and Security elements in a DDMS record. The purpose of these defaults is that many providers fail to deliver a classification and Owner/Producer with the metadata returned. These default settings are used if a metadata record is returned without security settings. **This feature can be turned on or off.**

- Open the web admin console.
 - <http://localhost:8181/system/console>
 - Username/Password: admin/admin
- Click on the Configuration tab.
- Find Catalog Security Defaults
- Select whether or not to apply these defaults by checking or unchecking the box marked "Apply Default Security Settings."

If checked, the default security settings specified in this configuration are applied to all records returned from the CDR-compliant Search Service that do not contain DDMS metadata. (If a metadata record is returned without DDMS, it also is without security markings). If unchecked, the records without the DDMS metadata are removed from the result set.

- If the applied defaults are selected, change the settings in the console to the default metadata security.
 - These settings can also be changed by editing the file `<INSTALL_DIRECTORY>/etc/ddf/ddf.DefaultSiteSecurity.cfg`
- Click Save at the bottom of the configuration window (or save the file).

- In the CDR Open Search Service, if one of the properties has a blank value, the "last-resort" default is U and USA.
- If the DefaultSiteSecurity.cfg file is deleted, it needs to be replaced otherwise all classification values are set to "last-resort" defaults U and USA.
- After the file is replaced, either restart DDF or the restart the CDR Open Search Service to reload the property values.

If this feature is turned off, the records from a CDR-compliant Search Service without DDMS metadata are dropped out of the result set. This means that a Max Results of 20 query parameters is specified and there are only 15 records containing DDMS metadata, the

result set only contains 15 records. The total count will accurately specify the number of records that match the query criteria in the Source that is queried.

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

Implementation Details

Imported Services

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

Exported Services

Registered Interface	Service Property	Value
ddf.catalog.source.FederatedSource		

Known Issues and Limitations

The OpenSearch Source does not provide a [Connected Source](#) interface.

DDF Catalog Solr

- [Solr Catalog Provider Apps](#) — implementation of a CatalogProvider using Apache Solr as the data store.
- [Standalone Solr Server App](#) — an Apache Solr instance as a Catalog data store within the distribution
 - [Solr Standalone Server Backup](#)

Solr Catalog Provider Apps

- [Description](#)
- [Usage](#)
 - [App Comparison / Usage Chart](#)
 - [When to Use](#)
- [Installation and Uninstallation](#)
 - [Embedded Solr Server and Solr Catalog Provider](#)
 - [Solr Catalog Provider for External Solr](#)
- [Configuration](#)
 - [Embedded Solr Server and Solr Catalog Provider](#)
 - [Solr Catalog Provider for External Solr](#)
- [Implementation Details](#)
 - [Indexing Text](#)
- [Known Issues](#)

Description

The Solr Catalog Provider (SCP) is an implementation of the `CatalogProvider` interface using Apache [Solr](#) as a data store. Some notable features of the SCP are

- Supports Extensible Metacards
- Fast, simple contextual searching
- [Indexes XML Attributes](#) as well as CDATA sections and XML text elements
- Simple relative (`//element`) and absolute pathing (`/root/element`) xpath support.
- Works with an embedded, local Solr Server (all-in-one Catalog)
 - No configuration necessary on a single-node Distribution
 - Data directory of solr indexes are configurable
- Works with a standalone Solr Server

Usage

The Solr Catalog Provider is used in conjunction with an Apache Solr Server data store. The Solr Catalog Provider can work with an embedded, local Solr Server instance or an external Solr Server. The embedded, local instance is a lightweight solution that works out of the box without any configuration. It however does not provide a [Solr Admin GUI](#) or a "REST-like HTTP/XML and JSON API." If that is necessary, see [Standalone Solr Server App](#).

Two different apps exist:

1. `catalog-solr-app` - includes the Solr Catalog Provider and an embedded Solr Server all-in-one.
2. `catalog-solr-external-app` - includes only the Solr Catalog Provider and is meant to be used only with the [Standalone Solr Server App](#) (`catalog-solr-server-app`).

App Comparison / Usage Chart

Feature	Embedded Solr		Standalone Solr	
	Pro	Con	Pro	Con
Scalability		<ul style="list-style-type: none">• Does not scale. Only runs one single server instance.• Does not allow the middle tier to be scaled.	<ul style="list-style-type: none">• Allows the middle-tier to be scaled by pointing various middle-tier instances to one server facade.• Possible data tier scalability with Solr Cloud. Solr Cloud allows for "high scale, fault tolerant, distributed indexing and search capabilities."	<ul style="list-style-type: none">• Solr Cloud Catalog Provider not implemented yet

Flexibility	<ul style="list-style-type: none"> • Can be embedded in Java easily. • Requires no HTTP Connection. • Uses the same interface as the Standalone Solr Server uses under the covers • Allows for full control over the Solr Server. No synchronous issues on startup, i.e. the Solr Server will synchronously start up with the Solr Catalog Provider • Runs within the same JVM • Setup and Installation is simple. "Unzip and run" 	<ul style="list-style-type: none"> • Only can be interfaced using Java 	<ul style="list-style-type: none"> • REST-like HTTP/XML and JSON APIs that make it easy to use from virtually any programming language • Ability to run in separate or same JVM of middle tier. 	
(Administrative) Tools	<ul style="list-style-type: none"> • External open source tools like Luke will work to allow admins to check index contents • JMX metrics reporting is enabled 	<ul style="list-style-type: none"> • No Admin Console. • No easy way to natively access (out of the box) what is in the index files or health of server at the data store level. 	<ul style="list-style-type: none"> • Contains Solr Admin GUI, which allows admins to query, check health, see metrics, see configuration files and preferences, etc • External open source tools like Luke will work to allow admins to check index contents • JMX metrics reporting is enabled 	
Security	<ul style="list-style-type: none"> • Does not open any ports which means no ports have to be secured. 		<ul style="list-style-type: none"> • Inherits app server security 	<ul style="list-style-type: none"> • Admin console must be secured and is openly accessible • REST-like HTTP/XML and JSON APIs must be secured • Current Catalog Provider implementation requires sending unsecured messages to Solr. Without a coded solution, requires network or firewall restrictions in order to secure.
Performance	<ul style="list-style-type: none"> • Requires no HTTP or Network overhead • Near Real-time indexing • Can understand complex queries 		<ul style="list-style-type: none"> • If scaled, high performance. • Near Real-time indexing 	<ul style="list-style-type: none"> • Possible network latency impact • Extra overhead when sent over HTTP. Extra parsing for XML, JSON, or other interface formats • Possible limitations upon requests and queries dependent on HTTP server settings
Backup/Recovery	<ul style="list-style-type: none"> • Can manually or through custom scripts back up the indexes 	<ul style="list-style-type: none"> • Must copy files when server is shutdown 	<ul style="list-style-type: none"> • Built-in Recovery tools which allow in-place backups (does not require server shutdown). • Backup of Solr indexes can be scripted 	<ul style="list-style-type: none"> • Recovery is done as a HTTP request

When to Use

Use the local, embedded Solr Catalog Provider when only one DDF instance is necessary and scalability is not an issue. The local, embedded Solr Catalog Provider requires no installation and little to no configuration since it ready out of the box. It is great for demonstrations, training exercises, or for sparse querying and ingesting. For heavy querying and ingesting processing, use the Standalone Solr Server on a separate machine. See the [Standalone Solr Server Recommended Configuration](#). Both Apps can store the same amount of data and indexes.

Installation and Uninstallation

The Solr Source can be installed and uninstalled using the normal processes described in the [Configuration](#) section. Ensure that no other Catalog Provider is installed before installing this Catalog Provider.

Embedded Solr Server and Solr Catalog Provider

Users can use the Solr Catalog Provider with an embedded Solr Server by installing (if it is not already installed) the feature, `catalog-solr-provider`. By installing this feature, it will install a Solr Catalog Provider and start up an instance of an embedded Java Solr Server within the distribution. Optional configurations are available. See the Configuration section for more information.

Solr Catalog Provider for External Solr

If the Solr Server is not embedded within the current distribution, a user will need to install the external Solr Catalog Provider by installing the feature `catalog-solr-external-provider`. This will not install any Solr Servers. Installing the feature will provide a user an "unconfigured" Solr Catalog Provider. See the Configuration section for how to configure this Solr Catalog Provider to connect to an external Solr Server.

Configuration

Embedded Solr Server and Solr Catalog Provider

No configuration is necessary in order for the embedded Solr Server and the Solr Catalog Provider to work out of the box. The standard installation described above is sufficient. When the `catalog-solr-provider` feature is installed, it by default stores the Solr index files to **<DISTRIBUTION_INSTALLATION_DIRECTORY>/data/solr**. A user does not have to specify any parameters. In addition, the `catalog-solr-provider` feature contains all files necessary for Solr to start the server.

However, this component can be configured to specify the directory to use for data storage using the normal processes described in the [Configuration](#) section.

The configurable properties for the SCP are accessed from the *Catalog Embedded Solr Catalog Provider* Configurations in the Admin Console.

Handy Tip

The Embedded (Local) Solr Catalog Provider works on startup without any configuration because a local embedded Solr Server is automatically started and pre-configured.

Configurable Properties

Title	Property	Type	Description	Default Value	Required
Data Directory File Path	dataDirectoryPath	String	<p>Specifies the directory to use for data storage. A shutdown of the server is necessary for this property to take effect. If a filepath is provided with directories that don't exist, SCP will attempt to create those directories. Out of the box (without configuration), the SCP writes to <DISTRIBUTION_INSTALLATION_DIRECTORY>/data/solr</p> <p>If <code>dataDirectoryPath</code> is left blank (empty string), it will default to <DISTRIBUTION_INSTALLATION_DIRECTORY>/data/solr.</p> <p>If <code>Data Directory File Path</code> is a relative string, the SCP will write the data files starting at the installation directory. For instance if the string <code>scp/solr_data</code> is provided, then the data directory would be at <DISTRIBUTION_INSTALLATION_DIRECTORY>/scp/solr_data</p> <p>If <code>Data Directory File Path</code> is <code>/solr_data</code> in Windows, the Solr Catalog Provider will write the data files starting at the beginning of the drive such as <code>C:/solr_data</code>.</p> <p>It is recommended to use an absolute filepath to minimize confusion such as <code>/opt/solr_data</code> in Linux or <code>C:/solr_data</code> in Windows. Permissions are necessary to write to the directory.</p>		No
Force Auto Commit	forceAutoCommit	Boolean / Checkbox	WARNING: Performance Impact. Only in special cases should auto-commit be forced. Forcing auto-commit makes the search results visible immediately.		No

Solr Configuration Files

The Apache Solr product has [Configuration](#) files to customize behavior for the Solr Server. These files can be found at **<DISTRIBUTION_INSTALLATION_DIRECTORY>/etc/solr**. Care must be taken in editing these files because they will directly affect functionality and performance of the Solr Catalog Provider. **A restart of the distribution is necessary for changes to take effect.**

Note on Solr Configuration File Changes

Solr Configuration files should not be changed in most cases. Changes to the `schema.xml` will most likely need code changes within the Solr Catalog Provider.

Moving Solr Data to a New Location

If SCP has been installed for the first time, then changing the (1) `Data Directory File Path` property and (2) restarting the distribution is all that is necessary because no data had been written into Solr previously. Nonetheless, if a user needs to change the location after the user has already ingested data in a previous location, these are the steps that are required:

1. Change the `Data Directory File Path` property within the *Catalog Embedded Solr Catalog Provider* Configuration in the Admin Console to the desired future location of the Solr data files.
2. [Shutdown the distribution](#).
3. Find the future location on the drive. If the current location does not exist, create the directories.
4. Find the location of where the current Solr data files exist and copy all the directories in that location to the future the location. For instance if the previous Solr data files existed at `C:/solr_data` and it is necessary to move it to `C:/solr_data_new`, then copy all directories within `C:/solr_data` into `C:/solr_data_new`. Usually this consists of copying the *index* and *tlog* directories into the new data directory.
5. Start the distribution. SCP should recognize the index files and be able to query them as it could before.

Note: Changes Require a Distribution Restart

If the `Data Directory File Path` property is changed, no changes will occur to the SCP until the distribution has been restarted.

Handy Tip

If `Data Directory File Path` property is changed to a new directory and the previous data is not moved into that directory, then no data will be in Solr. Solr will create an empty index. Therefore it is possible to have multiple places where Solr files are stored and a user can toggle between those locations for different sets of data.

Solr Catalog Provider for External Solr

In order for the external Solr Catalog Provider to work, it must be pointed at the external Solr Server. When the `catalog-solr-external-provider` feature is installed, it is in an unconfigured state until the user provides an HTTP url to the external Solr Server. The configurable properties for this SCP are accessed from the *Catalog External Solr Catalog Provider* Configurations in the Admin Console.

Configurable Properties

Title	Property	Type	Description	Default Value	Required
HTTP URL	url	String	HTTP URL of the standalone, preconfigured Solr 4.x Server.	http://localhost:8181/solr	Yes
Force Auto Commit	forceAutoCommit	Boolean / Checkbox	WARNING: Performance Impact. Only in special cases should auto-commit be forced. Forcing auto-commit makes the search results visible immediately.	Unchecked/False	No

Implementation Details**Indexing Text**

When storing fields, the Solr Catalog Provider will analyze and tokenize the text values of `STRING_TYPE` and `XML_TYPE` [AttributeTypes](#). These types of fields are indexed in at *least* three ways: in raw form, analyzed with case sensitivity, and then analyzed without concern to case sensitivity. Concerning XML, the Solr Catalog Provider will analyze and tokenize XML CDATA sections, XML Element text values, and XML Attribute values.

Known Issues

- When searching with the `ANY_TEXT` field, SCP does not search all text fields within the Catalog Provider. Instead, it searches the `METADATA` field.
- SCP does not fully support spatial capabilities.
- SCP does not support ingesting or querying `GeometryCollection WKT`.
- SCP does not support crossing the International Date Line or pole wrapping.
- SCP ignores the following `AttributeDescriptor` methods: `isIndexed`, `isTokenized`, `isMultivalued`, `isStored`. SCP instead indexes, tokenizes, and stores data based on the `AttributeFormat`, such as it will store and not index all fields labeled as `AttributeFormat.BINARY` regardless of user instruction. SCP as of now has no multivalue support even though it is supported by Solr.
- SCP has a 1000 nautical mile limit for nearest neighbor queries. If a point is not provided, then the centroid of the shape will be used for distance calculations.
- SCP does not support full `TextPath`. Attributes and equality expressions are not supported currently.

Standalone Solr Server App

- [Description](#)
- [Usage](#)
- [Installation](#)
- [Configuration](#)
 - [Recommended Standalone Configuration](#)
- [Known Issues](#)

Description

The Standalone Solr Server app gives the user an ability to run an Apache Solr instance as a Catalog data store within the distribution. The Standalone Solr Server app contains a Solr Web Application Bundle and preconfigured Solr configuration files. A Solr Web Application Bundle is essentially the Apache Solr war repackaged as a bundle and configured for use within this distribution.

Usage

Users can use this app to create a data store. Users would use this style of deployment over an embedded Java Solr Server when the user wants to install a Solr Server on a separate, dedicated machine for the sole purpose of isolated data storage or ease of maintenance. The Standalone Solr Server can now run in its own JVM (separate from endpoints and other frameworks) and accept calls with its "REST-like HTTP/XML and

JSON API."

This Standalone Solr Server app is meant to be used in conjunction with the [Solr Catalog Provider for External Solr](#). The Solr Catalog Provider acts as a client to the Solr Server.

Installation

This application's feature, `catalog-solr-server`, can be installed and uninstalled using the normal processes described in the Administrator's Guide's [Configuration](#) section. This feature is included out of the box in the current distribution. Installing the feature will copy the Solr configuration files in the distribution home directory and then deploy the configured Solr war. You can test that the server started correctly by visiting the Solr Admin interface at <http://localhost:8181/solr>.

Configuration

This application comes pre-configured to work with [Solr Catalog Provider Apps](#) implementations. For most use cases, no other configuration to the Solr Server is necessary with the standard distribution.

Recommended Standalone Configuration

In production environments, it is recommended that Solr Server App be run in isolation on a separate machine in order to maximize the Solr Server performance and use of resources such as RAM and CPU cores. The Standalone Solr Server, as its name suggests, does not require or depend on other apps such as the Catalog API, nor does it require their dependencies either such as Camel, CXF, etc. Therefore it is recommended to have the Solr Server app run on a lightweight DDF distribution such as the DDF Distribution Kernel. If clustering is necessary, the Solr Server App can run alongside the Platform App for clustering support.

Recommended Steps to Run a Standalone Solr Server (No data clustering)

1. Obtain and unzip the DDF Kernel (`ddf-distribution-kernel-<VERSION>.zip`).
2. [Start the distribution](#).
3. When the Kernel has loaded up with the DDF logo at the command prompt, execute

```
la
```

which is short for "list all". Verify that all bundles are `Active`.

Note on Lightweight Kernel

Since the kernel does not include all apps, if you were to do a "list" instead of "la," a very small number of results would be returned at this point.

4. Next install the `war` feature by executing the command

```
features:install war
```

No output will be shown, but running the `la` command again will show that Jetty and Pax Web bundles were installed.

5. Finally, deploy the Standalone Solr Server App by copying the `catalog-solr-server-app-<VERSION>.kar` into the `<DISTRIBUTION_HOME>/deploy` directory. Verify the Solr Server is up by checking the Solr Admin Console. If on the same server, the Solr Admin Console is located at <http://localhost:8181/solr>.

Known Issues

The standalone Solr Server fails to install if it has been previously uninstalled prior to the distribution being restarted.

Solr Standalone Server Backup

Solr Standalone Server Metadata Catalog Backup

The Need for a Backup & Recovery Plan

Prior to setting up backup for the Solr Metadata catalog, it is important to plan how backup and recovery will be executed. The amount and velocity of data entering the catalog differ depending on the use of the system. With this, there will be varying plans depending on the need. It is important to get a sense of how often the data changes in the catalog in order to determine how often the data should be backed up. When

something goes wrong with the system and data is corrupted, how much time is there to recover? A plan must be put in place to remove corrupted data from the catalog and replace it with backed up data in a time span that fits deadlines. Equipment must also be purchased to maintain backups, and this equipment may be co-located with local production systems or remotely located at a different site. A backup schedule will also have to be determined so that it does not affect end users interacting with the production system.

Backing Up Data from the Solr Server Standalone Metadata Catalog

The Solr server contains a built-in backup system capable of saving full snapshot backups of the catalog data upon request. Backups are created by using a web based service. Through making a web based service call utilizing the web browser, a timestamped backup can be generated and saved to a local drive, or location where the backup device has been mounted.

The URL for the web call contains three parameters that allow for the customization of the backup:

- **command:** allows for the command 'backup' to backup the catalog
- **location:** allows for a file system location to place the backup to be specified
- **numberToKeep:** allows the user to specify how many backups should be maintained. If the number of backups exceed the "numberToKeep" value, the system will replace the newest backup with the oldest one.

An example URL would look like the following:

http://127.0.0.1:8181/solr/replication?command=backup&location=d:/solr_data&numberToKeep=5

The IP address and port in the URL should be replaced with the IP address and port of the Solr Server. The following URL would run a backup, save the backup file in file location *D:/solr_data*, and it would keep up to (5) backup files at any time. To execute this backup, first ensure that the Solr server is running. Once the server is running, create the URL and copy it into a web browser window. Once the URL is executed, the following information is returned to the browser:

Solr Backup Response

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">15</int>
  </lst>
  <str name="status">OK</str>
</response>
```

If the status equals 0, then that means there was success. Qtime shows the time it took to execute the backup (in milliseconds). Backup files are saved in directories which are given the name snapshot along with a timestamp. Within the directory are all of the files that contain the data from the catalog.

Restoring Data to the Solr Server Standalone Metadata Catalog

When data has been corrupted or information has accidentally been deleted, a restoration of the catalog must occur. The backup files acquired from the previous section will be used to restore data into the catalog.

1. The first step in the process is to choose which data backup will be used for restoring the catalog. A most recent backup maybe the correct choice, or the last stable backup may be a better option.
2. At this point one more backup may be executed to save the corrupted data just in case it needs to be revisited.
3. The next step is to shutdown the Solr server. The restoring of the catalog cannot occur while the server is running.
4. Next step is to find the index which contains all of the Solr data. This index is found at *\$DDF_INSTALL/solr/collection1/data/index*. All of these files within the index directory should be deleted.

5. Now copy the files from the chosen backup directory into the index directory.
6. Restart the Solr server and the data should now be restored.

Suggestions For Managing Backup & Recovery

Here are some helpful suggestions for setting up data backups and recoveries:

- Acquire a backup drive that is separate from the media that runs the server. Mount this drive as a directory and save backups to that location.
- Ensure that the backup media has enough space to support the number of backups that need to be saved.
- Run a scheduler program that calls the backup URL on a timed basis.
- Put indicators in place that can detect when data corruption may have occurred.
- Testing a backup before recovery is possible. A replicated "staging" Solr server instance can be stood up, and the backup can be copied to that system for testing before moving it to the "production" system.

DDF Catalog KML

- [KML Query Response Transformer](#) — transforms a Query Response into a KML formatted document
- [KML Metacard Transformer](#) — transforms a Metacard into a KML formatted document
- [KML Network Link Endpoint](#) — allows a user to generate a View-based KML Query Results Network Link that can be opened with Google Earth, establishing a dynamic connection between Google Earth and DDF

KML Query Response Transformer

- [Description](#)
- [Usage](#)
 - [Example Output](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Implementation Details](#)
 - [Query Result Mapping](#)
- [Known Issues](#)

Description

The KML Query Response Transformer is responsible for translating a Query Response into a KML formatted document.

Usage

Using the [OpenSearch Endpoint](#) for example, query with the format option set to the KML shortname: `kml`.

```
http://localhost:8181/services/catalog/query?q=schematypesearch&format=kml
```

Example Output

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<kml xmlns:ns2="http://www.google.com/kml/ext/2.2"
xmlns="http://www.opengis.net/kml/2.2"
xmlns:ns4="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0"
xmlns:ns3="http://www.w3.org/2005/Atom">
  <Folder id="2c19380f-529f-4a91-867e-a7debc914244">
    <name>DDF Query Results</name>
    <open xsi:type="xs:boolean" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</open>
    <Document id="ef6335a1e79d46bba65d11cdd7d02c92-doc">
      <name>DDF Search</name>
      <open xsi:type="xs:boolean" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</open>
      <Style id="defaultStyle">
        <LineStyle>
```



```

        <color>8fff0000</color>
        <width>4.0</width>
    </LineStyle>
    <PolyStyle>
        <color>7dcc0000</color>
        <fill xsi:type="xs:boolean"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</fill>
        <outline xsi:type="xs:boolean"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</outline>
    </PolyStyle>
</Style>
<Placemark id="ef6335a1e79d46bba65d11cdd7d02c92">
    <name>DDF search</name>
    <description>A DDF Search found several results.
    <a
href="http://localhost:8181/services/catalog//ef6335a1e79d46bba65d11cdd7d0
2c92">
        DDF Search
    </a>
</description>
<styleUrl>#defaultStyle</styleUrl>
<Polygon>
    <outerBoundaryIs>
        <LinearRing>
            <coordinates>44.0,34.0 44.0,33.0 45.0,33.0 45.0,34.0
44.0,34.0</coordinates>
        </LinearRing>
    </outerBoundaryIs>
</Polygon>
</Placemark>
</Document>
<Document id="22e9daleefa74eb9b9f97636cab3f493-doc">
    <name>Brazil Amazon Rain Forest schematypesearch</name>
    <open xsi:type="xs:boolean" xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</open>
    <Style id="defaultStyle">
        <LineStyle>
            <color>8fff0000</color>
            <width>4.0</width>
        </LineStyle>
        <PolyStyle>
            <color>7dcc0000</color>
            <fill xsi:type="xs:boolean"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</fill>
            <outline xsi:type="xs:boolean"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</outline>
        </PolyStyle>
    </Style>

```

```

    <Placemark id="22e9daleefa74eb9b9f97636cab3f493">
      <name>Brazil Amazon Rain Forest schematypesearch</name>
      <description>
        <a
href="http://localhost:8181/services/catalog//22e9daleefa74eb9b9f97636cab3
f493">
          Brazil Amazon Rain Forest schematypesearch
        </a>
      </description>
      <styleUrl>#defaultStyle</styleUrl>
    </Placemark>
  </Document>
<Document id="ab3a5626ed80439aa8e2497835b54187-doc">
  <name>Position Predator Iraq Mission M2 10 schematypesearch</name>
  <open xsi:type="xs:boolean" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</open>
  <Style id="defaultStyle">
    <LineStyle>
      <color>8fff0000</color>
      <width>4.0</width>
    </LineStyle>
    <PolyStyle>
      <color>7dcc0000</color>
      <fill xsi:type="xs:boolean"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</fill>
      <outline xsi:type="xs:boolean"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</outline>
    </PolyStyle>
  </Style>
  <Placemark id="ab3a5626ed80439aa8e2497835b54187">
    <name>Book Search</name>
    <description>
      <a
href="http://localhost:8181/services/catalog//ab3a5626ed80439aa8e2497835b5
4187">
        Book Search
      </a>
    </description>
    <styleUrl>#defaultStyle</styleUrl>
  </Placemark>
</Document>
<Document id="d5c90fcc2be6471aaffa098262babf7e-doc">
  <name>Ballpark search</name>
  <open xsi:type="xs:boolean" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</open>
  <Style id="defaultStyle">
    <LineStyle>
      <color>8fff0000</color>
      <width>4.0</width>

```

```
</LineStyle>
<PolyStyle>
  <color>7dcc0000</color>
  <fill xsi:type="xs:boolean"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</fill>
  <outline xsi:type="xs:boolean"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</outline>
</PolyStyle>
</Style>
<Placemark id="d5c90fcc2be6471aaffa098262babf7e">
  <name>Ballpark search</name>
  <description>
    <a
href="http://localhost:8181/services/catalog//d5c90fcc2be6471aaffa098262ba
bf7e">
      Ballpark search
    </a>
  </description>
  <styleUrl>#defaultStyle</styleUrl>
</Placemark>
```

```
</Document>
</Folder>
</kml>
```

Installation and Uninstallation

Install the `catalog-kml-metacardtransformer` feature using the [Web Console](#) or [System Console](#).

Configuration

None

Implementation Details

Transformer Shortname	kml
MIME Type	application/vnd.google-earth.kml+xml

Query Result Mapping

Metacard	KML	Notes
Metacard.ID	Placemark ID	
Metacard.ID	Document ID	"-doc" suffix added
DDMS Title	Placemark Name	
DDMS Description & DDMS Title	Placemark Description	Placemark Description includes DDMS Description, the REST URL to access the complete metacard and the DDMS Title.
DDMS Geometry	KML Geometry	
	Folder ID	If the OpenSearch invocation query parameter <code>subscription</code> is specified, the Folder ID will be set to that. Otherwise, the Folder ID is randomly generated.

Known Issues

None

KML Metacard Transformer

- [Description](#)
- [Usage](#)
 - [Example Output](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Implementation Details](#)
 - [Metacard Mapping](#)
- [Known Issues](#)

Description

The KML Metacard Transformer is responsible for translating a Metacard into a KML formatted document.

Usage

Using the [REST Endpoint](#) for example, request a metacard with the transform option set to the KML shortname.

```
http://localhost:8181/services/catalog/96fa82e5c0034ef8aeaa45b15f126eef?transform=kml
```

Example Output

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<kml xmlns:ns2="http://www.google.com/kml/ext/2.2"
xmlns="http://www.opengis.net/kml/2.2"
  xmlns:ns4="urn:oasis:names:tc:ciq:xsd:schema:xAL:2.0"
xmlns:ns3="http://www.w3.org/2005/Atom">
  <Folder>
    <name>DDF Query Results</name>
    <open xsi:type="xs:boolean" xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</open>
    <Document id="96fa82e5c0034ef8aeaa45b15f126eef-doc">
      <name>Example Title</name>
      <open xsi:type="xs:boolean" xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</open>
      <Style id="defaultStyle">
        <LineStyle>
          <color>8fff0000</color>
          <width>4.0</width>
        </LineStyle>
        <PolyStyle>
          <color>7dcc0000</color>
          <fill xsi:type="xs:boolean"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</fill>
          <outline xsi:type="xs:boolean"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</outline>
        </PolyStyle>
      </Style>
      <Placemark id="96fa82e5c0034ef8aeaa45b15f126eef">
        <name>Example Title</name>
        <description>Example description.
          <a

href="http://localhost:8181/services/catalog//96fa82e5c0034ef8aeaa45b15f12
6eef">
            Example Title
          </a>
        </description>
        <styleUrl>#defaultStyle</styleUrl>
      </Placemark>
    </Document>
  </Folder>
</kml>
```

Installation and Uninstallation

Install the ddf-kml-transformer feature using the [Web Console](#) or [System Console](#).

Configuration

None

Implementation Details

Transformer Shortname	kml
MIME Type	application/vnd.google-earth.kml+xml

Metacard Mapping

Metacard	KML	Notes
Metacard.ID	Placemark ID	
Metacard.ID	Document ID	"-doc" suffix added
DDMS Title	Placemark Name	
DDMS Description & DDMS Title	Placemark Description	Placemark Description includes DDMS Description, the REST URL to access the complete metacard and the DDMS Title.
DDMS Geometry	KML Geometry	

Known Issues

None

KML Network Link Endpoint

- [Description](#)
- [Usage](#)
 - [Example Output](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Known Issues](#)

Description

The KML Network Link Endpoint allows a user to generate a View-based KML Query Results Network Link. This Network Link can be opened with Google Earth establishing a dynamic connection between Google Earth and DDF. The Network Link will perform a query against the [OpenSearch Endpoint](#) periodically based on the current view in the KML client. The query parameters for this query are obtained by a bounding box generated by Google Earth as well as any other query parameters specified in the request.

For instance, if an analyst is currently viewing Arizona in their Google Earth, the query results will only include results in Arizona. To generate this Network Link, make an HTTP GET request to `http://<DDF_HOST>:8181/services/catalog/kml`. Additionally, if an analyst only wanted to see query results matching the contextual parameter "Airport," they could specify `http://<DDF_HOST>:<DDF_PORT>/services/catalog/kml?q=Airport`. Now, the analyst, still viewing Arizona in Google Earth, would then receive all the catalog entries matching "Airport" in Arizona.

As an analyst changes their current view, the query will be re-executed with the bounding box of the new view. (This query gets re-executed 2 seconds after the user stops moving the view).

While looking at the same view, the query will be re-executed periodically to get all updates. The default refresh interval is 10 seconds. This can be specified as an "interval" query parameter in the Network Link request like this `http://<DDF_HOST>:<DDF_PORT>/services/catalog/kml?interval=5`. The "interval" parameter is specified in seconds.



Unknown macro: 'plantuml'

Usage

Once installed the KML Network Link Endpoint can be accessed at:

```
http://<DDF_HOST>:<DDF_PORT>/services/catalog/kml
```

Invoke the KML Network Link Endpoint using any OpenSearch query parameters except for geospatial parameters. The reason for this is that the current KML client view will provide the bounding box geospatial parameters for the query.

```
http://<DDF_HOST>:<DDF_PORT>/services/catalog/kml?q=Airport
```

Also, there is an optional query parameter "interval" that can be used to specify the refresh time in seconds (default is 10 seconds). With the View-based KML Query Results Network Link, Google Earth will refresh its results every <interval> time and every time the analyst stops moving the Google Earth view.

After the above request is sent, a KML Network Link document will be returned as a response to download or open. This KML Network Link can then be opened in Google Earth.

Example Output

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<kml xmlns="http://www.opengis.net/kml/2.2"
xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:gx="http://www.google.com/kml/ext/2.2"
xmlns:xal="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0">
  <Folder>
    <name>DDF OpenSearch Network Link</name>
    <open ns5:type="ns6:boolean"
xmlns:ns5="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns6="http://www.w3.org/2001/XMLSchema">true</open>
    <NetworkLink>

      <name>q=*&amp;src=local&amp;sort=date%3Adesc&amp;format=kml&amp;subscripti
on=8e566e5e-ec65-43ec-883f-f20f82c14fec</name>
      <open ns5:type="ns6:boolean"
xmlns:ns5="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns6="http://www.w3.org/2001/XMLSchema">true</open>
      <Link>

        <href>http://127.0.1.1:8181/services/catalog/query?q=*&amp;src=local&amp;s
ort=date%3Adesc&amp;format=kml&amp;subscription=8e566e5e-ec65-43ec-883f-f2
0f82c14fec</href>

        <refreshInterval>0.0</refreshInterval>
        <viewRefreshMode>onStop</viewRefreshMode>
        <viewRefreshTime>2.0</viewRefreshTime>
        <viewBoundScale>1.0</viewBoundScale>

      <viewFormat>bbox=[bboxWest],[bboxSouth],[bboxEast],[bboxNorth]</viewFormat>
      </Link>
    </NetworkLink>
  </Folder>
</kml>
```

Installation and Uninstallation

The KML Network Link Endpoint can be installed and uninstalled using the normal processes described in the [Configuration](#) section.

Configuration

This KML Network Link Endpoint has no configurable properties. It can only be installed or uninstalled.

Known Issues

- There has been a rarely occurring issue with the Network Link connection between Google Earth and DDF. Google Earth will stop

receiving updated responses. If this occurs, first try to delete the Network Link from Google Earth. If that does not resolve the issue, then restart DDF.

DDF Catalog Schematron

- **Schematron Pre-Ingest Plugin Framework** — provides a pre-ingest interceptor that validates the incoming request against a Schematron ruleset (or rule sets)

Schematron Pre-Ingest Plugin Framework

Description

The Schematron Validation Plugin (`plugin-schematron-validation` bundle) provides a pre-ingest interceptor that validates the incoming request against a Schematron ruleset (or rule sets). If the request has warnings or errors based on the Schematron validation, the request is marked as invalid and a SOAP fault is returned with details on the exact reason why the request was invalid. This bundle has the following characteristics:

- It provides the Schematron engine, meaning it provides the infrastructure to load, parse, and apply Schematron rule sets.
- It does not contain any Schematron ruleset(s) - those must be installed (as features) separately.

The Schematron validation bundle works with Schematron rule set bundles to obtain the rules for validation. The Schematron validation bundle and the Schematron rule set bundle are uninstalled by default. More information about Schematron in general can be found at <http://www.schematron.com>.

Understanding Schematron

Schematron is a language for making assertions about the presence or absence of patterns in XML documents. It is not a replacement for XML Schema (XSD) validation. Rather, it is used in conjunction with many grammar-based structure-validation languages, such as XSD.

Schematron is an ISO standard: ISO/IEC 19757-3:2006 Information technology -- Document Schema Definition Language (DSDL) -- Part 3: Rule-based validation -- Schematron

Schematron assertions are based on 2 simple actions:

- First, **find** context nodes in the document (typically an element) based on XPath criteria.
- Then, **check** to see if some other XPath expressions are true, for each of the nodes returned in the first step.

Schematron assertions (or rules) are defined in a `.sch` file by convention, which is an XML file conforming to Schematron's rules for defining assertions. This file is referred to as a "Schematron ruleset." These rules are contained in one `.sch` file, or a hierarchy of `.sch` files (`plugin-ddms-schematron-validation`). But there is ultimately one `.sch` file that includes or uses all of the other `.sch` files. This one `.sch` file is the "ruleset" used by the DDF Schematron Validation Service.

Schematron also includes SVRL (Schematron Validation Report Language) report generation, which is in XML format. This report includes the results of all of the Schematron rulesets' assertions, classifying them as warnings or errors (based on the ruleset).

DDF implements Schematron as a Pre-Ingest Plugin, running the Schematron ruleset(s) against each catalog entry in each create and update ingest request that DDF receives. The DDF Schematron Validation Pre-Ingest Plugin consists of 2 components: the Schematron "engine" and the client ruleset bundle(s). Each are described below.

Schematron Validation Plugin

The Schematron Validation Service is in a single OSGi bundle named `plugin-schematron-validation`. This bundle includes all of the code to implement:

- Loading and pre-compilation of the client ruleset bundle
- Executing the ruleset against ingest requests
- Generating the SVRL report. From this report, the Schematron Validation Service determines if errors and/or warnings were detected during validation. If errors or warnings exist, then validation fails and the ingest request is rejected. A SOAP fault is then returned to the client, including details on why the request is invalid.

The client's ruleset bundle determines what rules generate warnings and what rules generate errors. The Schematron Validation Service provides a configuration option (accessible via the Web Console's Configuration page) to suppress warnings. When this option is set, if only warnings are detected during Schematron validation, then the request is considered valid. By default, this suppress warnings option is unset (hence warnings result in invalid requests by default).

Validation is executed per catalog entry in the ingest request. Note that if multiple catalog entries are in the request, Schematron validation stops once a catalog entry is determined to be invalid. For example, if ten catalog entries are in a single create ingest request and entry #4 is invalid, entries 5 through 10 will not even be validated. Schematron returns an invalid status after entry #4 is validated.

Note that if only the Schematron Validation Service is installed, no Schematron validation occurs. This is because the Schematron Validation Service has no ruleset to validate the request against - it only provides the framework for Schematron rulesets to be applied to ingest requests. At least one (or more) client ruleset bundle must also be installed.

Schematron Client "Ruleset" Bundle(s)

A client must deploy at least one (or more) Schematron ruleset bundles before Schematron validation occurs.

The Schematron ruleset bundle consists of 3 required items:

- the `.sch` ruleset file defining the Schematron applied rules
- a bundle wiring specification file (e.g., Blueprint, Spring DM, Declarative Services, etc.) specifying the `.sch` file used and associating the ruleset to the Schematron Validation Service
- an OSGi metatype XML file that specifies the configurable options for the Schematron Validation Service (namely the suppress warnings option)

The diagram below illustrates how these Schematron components interact:



Unknown macro: 'plantuml'

Installing and Uninstalling

The Schematron Validation Library can be installed and uninstalled using the normal processes described in the [Configuration](#) section to install and uninstall the feature.

Configuration

There are no configuration options for this application.

Catalog Transformers

- **XSLT Transformer** — allows developers to create light-weight Query Response Transformers <https://wiki.macefusion.com/display/DDF/Query+Response+Transformer> and Metacard Transformers <https://wiki.macefusion.com/display/DDF/Metacard+Transformer> using only a bundle header and XSLT files.
- **Included Input Transformers** — convert input data into Catalog Metacards
 - **Tika Input Transformer** — transforms a Microsoft Office, PDF, and OpenOffice documents into a Catalog Metacard
 - **GeoJSON Input Transformer** — transforms GeoJSON into a Catalog Metacard
- **Included Metacard Transformers** — convert Metacards into other data formats
 - **HTML Metacard Transformer** — transforms a Metacard into an HTML formatted document
 - **XML Metacard Transformer** — transforms a Metacard into an XML formatted document
 - **GeoJSON Metacard Transformer** — transforms a Metacard into GeoJSON text
 - **Thumbnail Metacard Transformer** — retrieves the thumbnail bytes of a Metacard
 - **Metadata Metacard Transformer** — returns the Metacard.METADATA attribute value when given a Metacard.
 - **Resource Metacard Transformer** — retrieves the resource bytes of a Metacard product
- **Included Query Response Transformers** — convert Query Responses into other data formats
 - **Atom Query Response Transformer** — transforms a Query Response into an Atom 1.0 <http://tools.ietf.org/html/rfc4287> feed
 - **HTML Query Response Transformer** — transforms a Query Response into an HTML formatted document
 - **XML Query Response Transformer** — transforms a Query Response into an XML formatted document
 - **SearchUI** — a QueryResponseTransformer that not only provides results in a html format but also provides a convenient, simple querying user interface
- **Catalog Geo-formatter Library** — Geo library to help with conversion of geometry objects into various formats such as GeoJson, GeoRSS, etc.

XSLT Transformer

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.

Included Input Transformers

Input transformers convert input data into Catalog Metacards.

- **GeoJSON Input Transformer** — transforms GeoJSON into a Catalog Metacard
- **Tika Input Transformer** — transforms a Microsoft Office, PDF, and OpenOffice documents into a Catalog Metacard

Tika Input Transformer

- [Description](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Service Properties](#)
- [Implementation Details](#)
 - [DDMS to Metacard Mapping](#)
- [Known Issues](#)

Description

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 DDF Content Repository and the Metadata Catalog despite no DDMS metadata being generated.

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

Usage

Use the Tika Input Transformer if ingesting Microsoft documents, OpenOffice documents, or PDF documents into the DDF Content Repository and/or the Metadata Catalog is important, even if the metacard generated contains no DDMS metadata in it.

Installation and Uninstallation

Install the `catalog-transformer-tika` feature using the [Web Console](#) or [System Console](#). This feature is uninstalled by default.

Configuration

None

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	
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](#).
DDMS to Metacard Mapping

N/A

Known Issues

- The Tika Input Transformer does not create DDMS metadata, it just populates the metadata about metadata fields such as Creation Date, Modified Date, etc.

GeoJSON Input Transformer

- [Description](#)
- [Usage](#)
 - [Conversion](#)
 - [Metacard Extensibility](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Packaging Details](#)
 - [Feature Information](#)
 - [Included Bundles](#)
 - [Services](#)
 - [Exported Services](#)
- [Implementation Details](#)
 - [Exported Services](#)
- [Known Issues / Limitations](#)

Description

The GeoJSON Input Transformer is responsible for translating specific GeoJSON into a Catalog Metacard.

Usage

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"
http://localhost:8181/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.

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. Whitespace 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 Input Transformer supports custom, extensible properties on the incoming GeoJSON. It uses DDF's extensible metacard support to do this. 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.

Installation and Uninstallation

Install the catalog-rest-endpoint feature using the [Web Console](#) or [System Console](#).

Configuration

None

Packaging Details

Feature Information

N/A

Included Bundles

N/A

ServicesExported Services

ddf.catalog.transform.InputTransformer	
mime-type	application/json
id	geojson

Implementation Details

Exported Services

Registered Interface	Service Property	Value
ddf.catalog.transform.InputTransformer	mime-type	application/json

	id	geojson
--	----	---------

Known Issues / Limitations

Does not handle multiple geometries yet.

Included Metacard Transformers

Metacard transformers convert Metacards into other data formats.

- [HTML Metacard Transformer](#) — transforms a Metacard into an HTML formatted document
- [XML Metacard Transformer](#) — transforms a Metacard into an XML formatted document
- [GeoJSON Metacard Transformer](#) — transforms a Metacard into GeoJSON text
- [Thumbnail Metacard Transformer](#) — retrieves the thumbnail bytes of a Metacard
- [Metadata Metacard Transformer](#) — returns the Metacard.METADATA attribute value when given a Metacard.
- [Resource Metacard Transformer](#) — retrieves the resource bytes of a Metacard product

HTML Metacard Transformer

- [Description](#)
- [Usage](#)
 - [Example Output](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Implementation Details](#)
- [Known Issues](#)

Description

The HTML Metacard Transformer is responsible for translating a Metacard into an HTML formatted document.

Usage

Using the [REST Endpoint](#) for example, request a metacard with the transform option set to the HTML shortcode.

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

Example Output

Resource	identifier	qualifier	http://purl.org/dc/terms/URI			
		value	http://www.army.mil/-slideshows/2011/03/20/53540-usarpac-soldiers-support-operation-tomodachi/			
	title	classification	U			
		owner	ProducerUSA			
		USARPAC Soldiers support Operation Tomodachi				
	description	classification	U			
		owner	ProducerUSA			
		About 30 10th Support Group Soldiers deployed from Torii Station, Japan, to provide humanitarian aid to the citizens of Japan after an earthquake and tsunami crippled the country on March 11.				
	language	qualifier	http://www.ietf.org/rfc/rfc1766.txt			
		value	en-us			
	dates	infoCutOff	2011-03-19T21:00:00-07:00			
		posted	2011-03-19T21:00:00-07:00			
	publisher	classification	U			
			Organization			
			name	Asia and Pacific RSS Feed		
			name	Army Public Affairs fulfills the Army's obligation to keep the American people and the Army informed, and helps to establish the conditions that lead to confidence in America's Army and its readiness to conduct operations in peacetime, conflict and war.		
	format	Media	mimeType	text/html		
			extent	qualifier	Content-Length	
				value	14397	
				medium	digital	
subjectCoverage	Subject	category	labelNews			
geospatialCoverage	GeospatialExtent	boundingGeometry	Point	id	ID_1	
				srsName	EPSG:4326	
				pos	132.4833333	
					35.2166667	
security	classification	U				
	owner	ProducerUSA				

Installation and Uninstallation

Install the catalog-transformer-html feature using the [Web Console](#) or [System Console](#).

Configuration

None

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

XML Metacard Transformer

- [Description](#)
- [Usage](#)
 - [Example Output](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Implementation Details](#)
 - [Metacard to XML Mappings](#)
 - [AttributeTypes](#)
- [Known Issues](#)

Description

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.

Usage

Using the [REST Endpoint](#) for example, request a Metacard with the transform option set to the XML shortname.

```
http://localhost:8181/services/catalog/ac0c6917d5ee45bfb3c2bf8cd2ebaa67?transform=xml
```

Example Output

The schema file for the XML Metacard format, [metacard.xsd](#), is attached to this page.

```
<ns3:metacard ns1:id="ac0c6917d5ee45bfb3c2bf8cd2ebaa67"
xmlns:ns1="http://www.opengis.net/gml" xmlns:ns3="urn:catalog:metacard">
  <ns3:type>ddf.metacard</ns3:type>
  <ns3:source>ddf</ns3:source>
  <ns3:dateTime name="modified">
    <ns3:value>2013-01-29T17:09:19.980-07:00</ns3:value>
  </ns3:dateTime>
  <ns3:stringxml name="metadata">
    <ns3:value>
      <ddms:Resource
xmlns:ddms="http://metadata.dod.mil/mdr/ns/DDMS/2.0/"
xmlns:ICISM="urn:us:gov:ic:ism:v2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <ddms:identifier ddms:qualifier="http://example.test#URI"
ddms:value="http://example.test.html"/>
        <ddms:title ICISM:classification="U"
ICISM:ownerProducer="USA">Example Title</ddms:title>
        <ddms:description ICISM:classification="U"
ICISM:ownerProducer="USA">Example description.</ddms:description>
        <ddms:dates ddms:posted="2013-01-29"/>
        <ddms:rights ddms:copyright="true"
ddms:intellectualProperty="true" ddms:privacyAct="false"/>
        <ddms:creator ICISM:classification="U"
ICISM:ownerProducer="USA">
          <ddms:Person>
            <ddms:name>John Doe</ddms:name>
            <ddms:surname>Doe</ddms:surname>
          </ddms:Person>
        </ddms:creator>
        <ddms:subjectCoverage>
          <ddms:Subject>
            <ddms:category ddms:code="nitf" ddms:label="nitf"
ddms:qualifier="SubjectCoverageQualifier"/>
            <ddms:keyword ddms:value="schematypesearch"/>
          </ddms:Subject>
        </ddms:subjectCoverage>
        <ddms:temporalCoverage>
          <ddms:TimePeriod>
```



```
        <ddms:start>2013-01-29</ddms:start>
        <ddms:end>2013-01-29</ddms:end>
    </ddms:TimePeriod>
</ddms:temporalCoverage>
    <ddms:security ICISM:classification="U"
ICISM:ownerProducer="USA" />
    </ddms:Resource>
</ns3:value>
</ns3:stringxml>
<ns3:string name="resource-size">
    <ns3:value>N/A</ns3:value>
</ns3:string>
<ns3:geometry name="location">
    <ns3:value>
        <ns1:Point>
            <ns1:pos>2.0 1.0</ns1:pos>
        </ns1:Point>
    </ns3:value>
</ns3:geometry>
<ns3:dateTime name="created">
    <ns3:value>2013-01-29T17:09:19.980-07:00</ns3:value>
</ns3:dateTime>
<ns3:string name="resource-uri">
    <ns3:value>http://example.com</ns3:value>
</ns3:string>
<ns3:string name="metadata-content-type-version">
    <ns3:value>v2.0</ns3:value>
</ns3:string>
<ns3:string name="title">
    <ns3:value>Example Title</ns3:value>
</ns3:string>
<ns3:string name="metadata-content-type">
    <ns3:value>Resource</ns3:value>
</ns3:string>
<ns3:dateTime name="effective">
```

```
<ns3:value>2013-01-29T17:09:19.980-07:00</ns3:value>
</ns3:dateTime>
</ns3:metacard>
```

Installation and Uninstallation

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 using the [Web Console](#) or [System Console](#).

Configuration

None

Implementation Details

Metacard to XML Mappings

Metacard Variables	XML Element
id	metacard/@gml:id
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

GeoJSON Metacard Transformer

- [Description](#)
- [Usage](#)
 - [Example Output](#)
- [Installation and Uninstallation](#)

- [Configuration](#)
- [Implementation Details](#)
- [Known Issues](#)

Description

GeoJSON Metacard Transformer translates a Metacard into GeoJSON.

Usage

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

Unable to render {include}

 The included page could not be found.

Installation and Uninstallation

Install the `catalog-transformer-json` feature using the [Web Console](#) or [System Console](#).

Configuration

None

Implementation Details

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

Known Issues

Thumbnail Metacard Transformer

- [Description](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Implementation Details](#)
- [Known Issues](#)

Description

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

Usage

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`.

Installation and Uninstallation

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

Configuration

None

Implementation Details

Service Property	Value
id	thumbnail
shortname	thumbnail
mime-type	image/jpeg

Known Issues

None

Metadata Metacard Transformer

- [Description](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Implementation Details](#)

Description

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

Usage

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
```

Installation and Uninstallation

Catalog Transformers App will install this feature when deployed. This transformer's feature, `catalog-transformer-metadata`, can be uninstalled or installed using the normal processes described in the [Configuration](#).

Configuration

None

Implementation Details

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

Resource Metacard Transformer

- [Description](#)
- [Usage](#)
- [Configuration](#)
- [Implementation Details](#)
- [Known Issues](#)

Description

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

Usage

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)"/>
```

Installation and Uninstallation

This transformer is installed by installing the feature associated with the transformer "catalog-transformer-resource". To uninstall the transformer, you must uninstall the feature "catalog-transformer-resource".

Configuration

None

Implementation Details

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

Known Issues

None

Included Query Response Transformers

Query Response transformers convert Query Responses into other data formats.

- [Atom Query Response Transformer](#) — transforms a Query Response into an Atom 1.0 <http://tools.ietf.org/html/rfc4287> feed

- **HTML Query Response Transformer** — transforms a Query Response into an HTML formatted document
- **SearchUI** — a QueryResponseTransformer that not only provides results in a html format but also provides a convenient, simple querying user interface
- **XML Query Response Transformer** — transforms a Query Response into an XML formatted document

Atom Query Response Transformer

- Description
 - Sample Results
 - Query Result Mapping
- Usage
- Installation and Uninstallation

Description

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](#).

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/ee7a161e01754b9db1
872bfe39d1ea09" 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="4535c53fc8bc4404ald32a5ce7a29585">
  <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
/feed/author/name	Platform Global Configuration <code>organization</code>
/feed/generator	Platform Global Configuration <code>site name</code>
/feed/generator/@version	Platform Global Configuration <code>version</code>
/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. <i>Metacard.getSourceId()</i>
/feed/entry/relevance:score	Result's relevance score if applicable. <i>Result.getRelevanceScore()</i>
/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 <code>ddf.catalog.transform.MetacardTransformer</code> with <code>shortname=</code> If no transformer found, <code>/feed/entry/content/@type</code> will be <code>text</code> and Metacard.ID is displayed <div> <div>Sample Content with no Metacard Transformation</div> <pre> <content type="text">4e1f38d1913b4e93ac622e6c1b258f89</content> </pre> </div>

Usage

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 (See [Implementation Details](#) on the details to acquire the Service).

Installation and Uninstallation

Catalog Transformers App will install this feature when deployed. This transformer's feature, `catalog-transformer-atom`, can be uninstalled or installed using the normal processes described in the [Configuration](#) section.

HTML Query Response Transformer

- [Description](#)
- [Usage](#)
 - [Example Output](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Implementation Details](#)
 - [Metacard Result to HTML Table Column Mapping](#)
- [Known Issues](#)

Description

The HTML Query Response Transformer transforms metacards and results from queries into an html format that is easier to read and understand. For results it provides the following fields when available:

- Hyperlinked title
- Created date
- Hyperlinked Product information
- Spatial information
- Thumbnail



Usage

Using the [OpenSearch Endpoint](#) for example, query with the format option set to either the HTML shortname (`html`) or the HTML table shortname (`html-table`).

```
http://localhost:8181/services/catalog/query?q=Sarin&format=html
```

Example OutputHTML

1 results

[Local version - Roadside Bomb Releases Sarin Gas in Quait \(Internet Only\)](#) 

POINT (2.0 1.0)

on 2013-01-25T11:03:47.940-07:00

HTML Table

1 results

#	Title	Date	Product	Spatial	Thumbnail
1	Local version – Roadside Bomb Releases Sarin Gas in Quait	2013-01-25T11:03:47.940-07:00		POINT (2.0 1.0)	

Installation and Uninstallation

Install the `catalog-transformer-ui` feature using the [Web Console](#) or [System Console](#).

Configuration

None

Implementation Details

Registered Service	Service Property	Value
ddf.catalog.transform.QueryResponseTransformer	title	View as Html-table...
	shortname	html-table
	description	Transforms query results into html-table

Metacard Result	HTML Table Column	Notes
Metacard.TITLE	Title	
Metacard.CREATED	Date	
Metacard.RESOURCE_URI	Product	
Metacard.GEOGRAPHY	Spatial	
Metacard.THUMBNAIL	Thumbnail	if thumbnail available

Metacard Result to HTML Table Column Mapping

Known Issues

None

XML Query Response Transformer

- [Description](#)
- [Usage](#)
 - [Example Output](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Implementation Details](#)
- [Known Issues](#)

Description

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.

Usage

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>
```

Installation and Uninstallation

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 [Web Console](#) or [System Console](#).

Configuration

None

Implementation Details

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

See [XML Metacard Transformer](#) Implementation Details as to how Metacard Java object information is mapped into XML.

Known Issues

None

SearchUI

Description

The SearchUI is a `QueryResponseTransformer` that not only provides results in a 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.

Query Response Result Mapping

SearchUI Column Title	Catalog Result	Notes
Title	Metacard.TITLE	The title maybe hyperlinked to view the full Metacard
Source	Metacard.getSourceId()	Source where the Metacard was discovered
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

Sample Output

The screenshot shows the SearchUI interface. On the left is a sidebar with search criteria filters: KEYWORDS (with a text input containing 'ddf'), TIME (with buttons for Any, Relative, Absolute), LOCATION (with buttons for Any, Point-Radius, Bounding Box), TYPE (with buttons for Any, Specific Types), and ADDITIONAL SOURCES (with buttons for None, All Sources, Specific Sources). At the bottom of the sidebar are 'Search' and 'Clear' buttons. The main area on the right shows 'Total Results: 1' and a table with columns: Title, Source, Location, and Time. The table contains one row with the following data: Title is 'myTitle' (hyperlinked), Source is 'ddf.distribution', Location is 'POINT (30 10)', and Time is 'Received: 2013-05-24 12:45:14 MST'. A 'Back to top' link is visible at the bottom right of the results area.

Usage

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, clicking on the following link <http://localhost:8181/search> should bring up the SearchUI.

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.

Installation

Catalog Transformers App will install this feature when deployed. This transformer's feature, `catalog-transformer-ui`, can be uninstalled or installed using the normal processes described in the [Configuration](#) section.

Configuration

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

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

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*.

Catalog Geo-formatter Library

Geo library to help with conversion of geometry objects into various formats such GeoJson, GeoRSS, etc.

Used by catalog-transformer-json and catalog-transformer-atom.

Feature name: catalog-transformer-geoformatter.

DDF Content Applications

DDF Content Core

- [Content Cataloger Plugin](#) — provides the implementation to parse content, create a Metacard, and create/update/delete catalog entries in the Metadata Catalog.
- [Directory Monitor](#) — allows ApplicationName to monitor a directory for files to be ingested into the Content Framework
- [File System Storage Provider](#) — provides the implementation to create/update/delete content items as files in the DDF Content Repository.
- [Standard Content Framework](#) — provides the reference implementation of a Content Framework

Content Cataloger Plugin

- [Description](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Known Issues](#)

Description

The Content Cataloger Plugin provides the implementation to parse content, create a Metacard, and create/update/delete catalog entries in the Metadata Catalog. The Content Cataloger Plugin is an implementation of the [ContentPlugin](#) interface. When installed, it is invoked by the [Content Framework](#) after a content item has been processed by the [Storage Provider](#) but before the response is returned to the [Content Endpoint](#).

The Content Cataloger Plugin searches the OSGi service registry for all services registered as [InputTransformers](#) that can process the content item's mime type. If such a service is found, the service is invoked (for `create` and `update` operations; `delete` operations are handled internally by the Content Cataloger Plugin). The [InputTransformer](#) service accepts the content item's `InputStream` and parses it, creating a Metacard that is returned to the Content Cataloger Plugin. This Metacard is then used in the `create` and `update` operations invoked on the [Catalog Framework](#) to interface with the Metadata Catalog.

Details on how to develop an Input Transformer, with either Java or Apache Camel, can be found in the [Developing an Input Transformer](#) section of the DDF Developer's Guide.

Usage

Use the Content Cataloger Plugin if create/update/delete of catalog entries in the Metadata Catalog based on the content item are desired. These CUD operations on the Metadata Catalog are made possible by parsing the content item to create a Metacard and then using this Metacard in the CUD operations on the [Catalog Framework](#). The Content Cataloger Plugin is the only component in the DDF Content Framework that has the ability to interface with the [Catalog Framework](#) (and hence the Metadata Catalog).

Installation and Uninstallation

The Content Cataloger Plugin is bundled as the `content-core-catalogerplugin` feature and can be installed and uninstalled using the normal processes described in the [Configuration](#) section of the Administrator's Guide.

Configuration

There are no configurable properties for this component. This component can only be installed and uninstalled.

Known Issues

- Content Cataloger Plugin is only partially transactional. On create operations where the content is being stored in the content repository and the content is being parsed to generate a metacard for insertion into the Metadata Catalog, the content storage will be undone (i.e., the recently inserted content removed from the content repository) if the Metadata Catalog insertion encountered problems. Update and delete operations have no transactional capabilities, hence once the content is updated or deleted this cannot be undone. Therefore, the content repository and Metadata Catalog could get out of sync.

Directory Monitor

- [Description](#)
- [Usage](#)
 - [Sample Usage Scenarios](#)
 - [Scenario 1: Monitor single directory for storage and processing, with no file backup](#)
 - [Scenario 2: Monitor single directory for storage with file backup](#)
 - [Scenario 3: Monitor multiple directories for processing only, with file backup - errors encountered on some ingests](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
 - [Configuring Content Directory Monitors](#)
 - [Managed Service Factory PID](#)
 - [Configurable Properties](#)
- [Known Issues](#)

Description

The Content Directory Monitor allows files placed in a monitored directory to be ingested into the DDF Content Repository and/or the Metadata Catalog (MDC). A monitored directory is a directory configured to be polled by DDF periodically (typically every one second) for any new files added to the directory that should be ingested into the Content Framework.

The typical execution flow of the Directory Monitor is:

- A new file is detected in the monitored directory,
- The file's contents are passed on to the Content Framework and processed based on whether the monitored directory's processing directive was:
 - Configured to just store the file in the DDF Content Repository,
 - Configured to just process the file's metadata and ingest it into the MDC, or
 - Configured to both store the file in the Content Repository and ingest it into the MDC.
- If the response from the Content Framework is successful, indicating the content was stored and/or processed, then the file in the monitored directory is either deleted (default behavior), or copied to a sub-directory called `.ingested` (see below for how to configure this behavior). If the response from the Content Framework was unsuccessful or a failure occurred, then the file is moved from the monitored directory to a sub-folder named `.errors`, allowing easy identification of the ingested files that had problems.

Multiple monitored directories can be configured, each monitoring different directories.

Usage

The Content Directory Monitor provides the capability to easily create content in the DDF Content Repository and Metacards in the MDC by simply placing a file in a directory that has been configured to be monitored by DDF. For example, this would be useful for copying files from a hard drive (or directory) in a batch-like operation to the monitored directory and having all of the files processed by the Content Framework.

Sample Usage Scenarios

Scenario 1: Monitor single directory for storage and processing, with no file backup

- The Content Directory Monitor is configured with:
 - the **relative** path of `inbox` for the Directory Path,
 - Processing Directive set to Store and Process, and
 - the Copy Ingested Files is not checked
- As files are placed in the monitored directory `<DDF_INSTALL_DIR>/inbox`, the files are ingested into the Content Framework.
 - The Content Framework generates a GUID for the create request for this ingested file
 - Since the Store and Process directive was configured the ingested file is passed on to the Content File System Storage Provider which:
 - creates a sub-directory in the Content Repository using the GUID, and

- places the ingested file into this GUID sub-directory using the file name provided in the request
- The Content Framework then invokes the Catalog Content Plugin which:
 - looks up the Input Transformer associated with the ingested file's mime type
 - this Input Transformer creates a Metacard based on the ingested file's contents
 - invokes the Catalog Framework, which inserts the metacard into the MDC
- Content Framework sends back a successful status to the Camel route that was monitoring the directory
- Camel route completes and deletes the file from the monitored directory

Scenario 2: Monitor single directory for storage with file backup

- The Content Directory Monitor is configured with:
 - the **absolute** path of `/usr/my/home/dir/inbox` for the Directory Path,
 - Processing Directive set to Store only, and
 - the Copy Ingested Files is checked
- As files are placed in the monitored directory `/usr/my/home/dir/inbox`, the files are ingested into the Content Framework.
 - The Content Framework generates a GUID for the create request for this ingested file
 - Since the Store directive was configured the ingested file is passed on to the Content File System Storage Provider which:
 - creates a sub-directory in the Content Repository using the GUID, and
 - places the ingested file into this GUID sub-directory using the file name provided in the request
 - Content Framework sends back a successful status to the Camel route that was monitoring the directory
 - Camel route completes and moves the file from the monitored directory to its sub-directory `/usr/my/home/dir/inbox/.ingested`

Scenario 3: Monitor multiple directories for processing only, with file backup - errors encountered on some ingests

- Two different Content Directory Monitors are configured with:
 - the **relative** path of `inbox` and `inbox2` for the Directory Path,
 - Processing Directive on both directory monitors set to Process, and
 - the Copy Ingested Files is checked for both directory monitors
- As files are placed in the monitored directory `<DDF_INSTALL_DIR>/inbox`, the files are ingested into the Content Framework.
 - The Content Framework generates a GUID for the create request for this ingested file
 - Since the Process directive was configured the ingested file is passed on to the Catalog Content Plugin which:
 - looks up the Input Transformer associated with the ingested file's mime type, but no Input Transformer is found
 - an exception is thrown
 - Content Framework sends back a failure status to the Camel route that was monitoring the directory
 - Camel route completes and moves the file from the monitored directory to the `.errors` sub-directory
- As files are placed in the monitored directory `<DDF_INSTALL_DIR>/inbox2`, the files are ingested into the Content Framework.
 - The Content Framework generates a GUID for the create request for this ingested file
 - The Content Framework then invokes the Catalog Content Plugin which:
 - looks up the Input Transformer associated with the ingested file's mime type
 - this Input Transformer creates a Metacard based on the ingested file's contents
 - invokes the Catalog Framework, which inserts the metacard into the MDC
 - Content Framework sends back a successful status to the Camel route that was monitoring the directory
 - Camel route completes and moves the file from the monitored directory to its `.ingested` sub-directory

Installation and Uninstallation

The Content Directory Monitor is packaged as the `content-core-directorymonitor` feature and is part of the `content-core-app`. It is installed by default.

It can be installed and uninstalled using the normal processes described in the [Configuration](#) section of the Administrator's Guide.

Note that the `content-core-catalogerplugin` feature must be installed for the metacards to be created and inserted into the MDC. This feature provides the linkage between the [Content Framework](#) and the [Catalog Framework](#). If the client attempts a `STORE_AND_PROCESS` or a `PROCESS` only without this feature installed a failure will be returned.

Configuration

This component can be configured using the normal processes described in the Administrator's Guide's [Configuration](#) section.

The configurable properties for the Content Directory Monitor are accessed from the *Content Directory Monitor* Configuration in the Admin Console.

Configuring Content Directory Monitors

Managed Service Factory PID

`ddf.content.core.directorymonitor.ContentDirectoryMonitor`
Configurable Properties

Title	Property	Type	Description	Default Value	Required
-------	----------	------	-------------	---------------	----------

Directory Path	monitoredDirectoryPath	String	Specifies the directory to be monitored. Can be a fully-qualified directory or a relative path (which is relative to the DDF installation directory).	N/A	Yes
Processing Directive	directive	String	One of three possible values from a drop down box: Store only - indicates to only store content in Content Repository Process only - indicates to only create Metacard and insert into MDC Store and Process - do both	Store and Process	Yes
Copy Files to Backup Directory	copyIngestedFiles	Boolean	Checking this option indicates that a backup of the file placed in the monitored directory should be made upon successful processing of the file. The file is moved into the <code>.ingested</code> sub-directory of the monitored directory.	False	No

Known Issues

None

File System Storage Provider

- [Description](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Known Issues](#)

Description

The File System Storage Provider provides the implementation to create/update/delete content items as files in the DDF Content Repository. The File System Storage Provider is an implementation of the Storage Provider interface. When installed, it is invoked by the [Content Framework](#) to create, update, or delete a file in the DDF Content Repository.

For create operations, the File System Storage Provider (using the [MimeTypeMapper](#)) examines the mime type of the content item and determines the extension to use for the file to be stored. The File System Storage Provider also auto-generates a Globally Unique ID (GUID) for the content item. This GUID is used as the sub-directory for the content item's location in the Content Repository. This is to insure the files in the Content Repository are more evenly distributed rather than all being stored in one monolithic directory. The content is stored using the file name specified in the create request.

As an example, if the content item's mime type was `image/nitf`, then:

- the file extension would be `.nitf`
- a GUID would be auto-generated (an example GUID would be `54947df8-0e9e-4471-a2f9-9af509fb5889`)
- the file name is specified in the `create` request (example: `myfile.nitf`)
- the location in the Content Repository would be determined based on the GUID and the file name specified in the request (example: `54947df80e9e4471a2f99af509fb5889/myfile.nitf`)

For `read` operations, the File System Storage Provider reads the content file with the GUID specified in the `ReadRequest`.

For `update` operations, the File System Storage Provider updates the content file with the content item's new `InputStream` contents. The GUID of the content file to be updated is included in the `UpdateRequest`.

For `delete` operations, the File System Storage Provider deletes the content file with the GUID specified in the `DeleteRequest`.

A sub-directory is created for each entry in the content store, so there will be limitations based on the file system that is used, i.e., the maximum amount of sub-directories supported for a file system.

Usage

Use the File System Storage Provider if create/read/update/delete of content item's to a file system are desired.

Installation and Uninstallation

The File System Storage Provider is packaged as the `content-core-filesystemstorageprovider` feature and can be installed and uninstalled using the normal processes described in the [Configuration](#) section of the Administrator's Guide. This feature is installed by default.

Configuration

The location used for content storage can be configured in the webconsole under Configuration -> Content File System Storage Provider.

Known Issues

None

Standard Content Framework

The Standard Content Framework provides the reference implementation of a [Content Framework](#) that implements all requirements of the Content API. `ContentFrameworkImpl` is the implementation of the Standard Content Framework.

Usage

The Standard Content Framework is the core class of DDF Content. It provides the methods for read, create, update, and delete (CRUD) operations on the [Storage Provider](#).

Use this framework if:

- access to a storage provider to create, update, and delete content items in the DDF Content Repository are required
- the ability to parse content, create a Metacard, and then create, update, and delete catalog entries in the Metadata Catalog based on the parsed content are required

Installation and Uninstallation

The Standard Content Framework is bundled in the `content-core` feature and is part of the `content-core-app`. It can be installed and uninstalled using the normal processes described in the [Configuration](#) section.

Configuration

There are no configuration properties for this component. This component can only be installed and uninstalled.

Known Issues

None

DDF Content REST CRUD Endpoint

- [Description](#)
- [Usage](#)
 - [Interacting with REST endpoint](#)
 - [Create Request Multipart/Form-Data Parameters](#)
 - [Update and Delete Request HTTP Header Parameters](#)
 - [Sample Requests and Responses](#)
 - [cURL Commands](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Known Issues](#)

Description

The Content REST Endpoint allows clients to perform CRUD operations on the Content Repository using REST, a simple architectural style that performs communication using HTTP.

The URL exposing the REST functionality will be located at `http://<DDF_HOST>:<DDF_PORT>/services/content`, where `DDF_HOST` is the IP address of where DDF is installed and `DDF_PORT` is the port number DDF is listening on.

The Content REST CRUD Endpoint provides the capability to read, create, update, and delete content in the Content Repository as well as create, update, and delete Metacards in the catalog provider, i.e., the Metadata Catalog (MDC). Furthermore, this endpoint allows the client to perform the create/update/delete operations on just the Content Repository, or just the MDC, or both in one operation.

The Content Framework is currently transactional **only** for create operations. Therefore, if the client sends a create request to create content in the DDF Content Repository and also process the content to create a Metacard, and ingest it into the MDC (i.e., `directive=STORE_AND_PROCESS`), if a problem is encountered during the catalog ingest the content is removed from the DDF Content Repository, analogous to a rollback. This is so that the DDF Content Repository and the MDC are kept in sync.

The Content Framework does **not** support rollback capability for update or delete operations that affect both the DDF Content Repository and the MDC.

Usage

The Content REST CRUD Endpoint provides the capability to read, create, update, and delete content in the DDF Content Repository as well as create, update, and delete Metacards in the catalog provider as follows. Sample requests and responses

are provided in a separate table.

Operation	HTTP Request	Details	Example URL
Create Content and Catalog Entry	HTTP POST	<p>The multipart/form-data REST request that contains the binary data to be stored in the DDF Content Repository and to be parsed to create a Metacard for ingest into the MDC. This binary data can be included in the request's body or as a file attachment.</p> <p>An HTTP 201 CREATED status code is returned to the client with:</p> <ul style="list-style-type: none"> Content-ID HTTP header set to GUID assigned to content item by Content Framework Catalog-ID HTTP header set to the catalog ID assigned to the new catalog entry created based on the Metacard generated from the parsed content Content-URI HTTP header set to the resource URI for the content stored in the DDF Content Repository Location URI HTTP header with URI containing the content ID 	<p><code>http://<DDF_HOST>:<DDF_PORT>/services/content</code></p> <p>where the <code>directive</code> form parameter is set to <code>STORE_AND_PROCESS</code></p> <p>and the <code>file</code> form parameter specifying the binary data, with an optional <code>filename</code> parameter that specifies the name of the file the content should be stored as in the DDF Content Repository.</p>
Create Content Only	HTTP POST	<p>The multipart/form-data REST request that contains the binary data to be stored in the DDF Content Repository. This binary data can be included in the request's body or as a file attachment.</p> <p>An HTTP 201 CREATED status code is returned to the client with:</p> <ul style="list-style-type: none"> Content-ID HTTP header set to GUID assigned to content item by Content Framework Location URI HTTP header with URI containing the content ID 	<p><code>http://<DDF_HOST>:<DDF_PORT>/services/content</code></p> <p>where the <code>directive</code> form parameter is set to <code>STORE</code></p> <p>and the <code>file</code> form parameter specifying the binary data, with an optional <code>filename</code> parameter that specifies the name of the file the content should be stored as in the DDF Content Repository.</p>
Create Catalog Entry Only	HTTP POST	<p>The multipart/form-data REST request that contains the binary data to be parsed to create a Metacard for ingest into the MDC. This binary data can be included in the request's body or as a file attachment.</p> <p>An HTTP 200 OK status code is returned to the client with:</p> <ul style="list-style-type: none"> Catalog-ID HTTP header set to the catalog ID assigned to the new catalog entry created based on the Metacard generated from the parsed content 	<p><code>http://<DDF_HOST>:<DDF_PORT>/services/content</code></p> <p>where the <code>directive</code> form parameter is set to <code>PROCESS</code></p> <p>and the <code>contentUri</code> form parameter is set to the URI of content being processed</p> <p>and the <code>file</code> form parameter specifying the binary data.</p>
Update Content and Catalog Entry	HTTP PUT	<p>The ID of the content item in the DDF Content Repository to be updated is appended to the end of the URL.</p> <p>The body of the REST request contains the binary data to update the DDF Content Repository.</p> <p>An HTTP 200 OK status code is returned to the client with:</p> <ul style="list-style-type: none"> Content-ID HTTP header set to GUID updated by the Content Framework Catalog-ID HTTP header set to the catalog ID that was updated in the MDC 	<p><code>http://<DDF_HOST>:<DDF_PORT>/services/content/ABC123</code></p> <ul style="list-style-type: none"> where <code>ABC123</code> is the ID of the content item to be updated the <code>directive</code> HTTP header parameter is set to <code>STORE_AND_PROCESS</code>

Update Content Only	HTTP PUT	<p>The ID of the content item in the DDF Content Repository to be updated is appended to the end of the URL.</p> <p>The body of the REST request contains the binary data to update the DDF Content Repository.</p> <p>An HTTP 200 OK status code is returned to the client with:</p> <ul style="list-style-type: none"> Content-ID HTTP header set to GUID updated by the Content Framework 	<p><code>http://<DDF_HOST>:<DDF_PORT>/services/content/ABC123</code></p> <ul style="list-style-type: none"> where ABC123 is the ID of the content item to be updated the directive HTTP header parameter is set to <code>STORE</code>
Update Catalog Entry Only	HTTP PUT	<p>The URI of the content item in the MDC to be updated is specified in the <code>contentUri</code> HTTP header parameter.</p> <p>The body of the REST request contains the binary data to update the catalog entry in the MDC.</p> <p>An HTTP 200 OK status code is returned to the client with:</p> <ul style="list-style-type: none"> Catalog-ID HTTP header set to the catalog ID that was updated in the MDC 	<p><code>http://<DDF_HOST>:<DDF_PORT>/services/content</code></p> <ul style="list-style-type: none"> the <code>contentUri</code> HTTP header parameter is set to the URI of the catalog entry in the MDC to be updated
Delete Content and Catalog Entry	HTTP DELETE	<p>The ID of the content item in the DDF Content Repository to be deleted is appended to the end of the URL.</p> <p>HTTP status code of 204 NO CONTENT is returned upon successful deletion.</p> <ul style="list-style-type: none"> Content-ID HTTP header set to GUID deleted by the Content Framework Catalog-ID HTTP header set to the catalog ID that was deleted from the MDC 	<p><code>http://<DDF_HOST>:<DDF_PORT>/services/content/ABC123</code></p> <ul style="list-style-type: none"> where ABC123 is the ID of the content item to be deleted the directive HTTP header parameter is set to <code>STORE_AND_PROCESS</code>
Delete Content Only	HTTP DELETE	<p>The ID of the content item in the DDF Content Repository to be deleted is appended to the end of the URL.</p> <p>HTTP status code of 204 NO CONTENT is returned upon successful deletion.</p>	<p><code>http://<DDF_HOST>:<DDF_PORT>/services/content/ABC123</code></p> <ul style="list-style-type: none"> where ABC123 is the ID of the content item to be deleted the directive HTTP header parameter is set to <code>STORE</code>
Delete Catalog Entry Only	HTTP DELETE	<p>The URI of the content item in the MDC to be deleted is specified in the <code>contentUri</code> HTTP header parameter.</p> <p>HTTP status code of 204 NO CONTENT is returned to the client upon successful deletion with:</p> <ul style="list-style-type: none"> Catalog-ID HTTP header set to the catalog ID that was deleted from the MDC 	<p><code>http://<DDF_HOST>:<DDF_PORT>/services/content</code></p> <ul style="list-style-type: none"> the <code>contentUri</code> HTTP header parameter is set to the URI of the catalog entry in the MDC to be deleted
Read	HTTP GET	<p>The ID of the content item in the DDF Content Repository to be retrieved is appended to the end of the URL.</p> <p>An HTTP 200 OK status code is returned upon successful read and the contents of the retrieved content item are contained in the HTTP body.</p>	<p><code>http://<DDF_HOST>:<DDF_PORT>/services/content/ABC123</code> where ABC123 is the ID of the content item to be retrieved</p>

Note that for all Content REST CRUD commands only one content item ID is supported in the URL, i.e., bulk operations are not supported.

Interacting with REST endpoint

Any web browser can be used to perform a REST read. Various other tools and libraries can be used to perform the other HTTP operations on the REST endpoint (e.g., soapUI, cURL, etc.)

Create Request Multipart/Form-Data Parameters

The `create` (HTTP POST) request is a multipart/form-data request, allowing the binary data (i.e., the content) to be either included in the request's body or attached as a file. This binary data is defined in a `Content-Disposition` part of the request where the `name` parameter is set to `file` and the optional `filename` parameter indicates the name of the file that the content should be stored as.

Optional form parameters for the `create` request are the `directive` and `contentUri`. The `directive` form parameter's value can be either `STORE`, `PROCESS`, or `STORE_AND_PROCESS`, indicating if the content should be only stored in the Content Repository, only processed to generate a Metacard and then ingested into the MDC, or both. the `directive` form parameter will default to `STORE_AND_PROCESS` if it is not specified.

The `contentUri` form parameter allows the client to specify the URI of a product stored remotely/externally (relative to DDF). This `contentUri` is used to indicate that the client will manage the content storage but wants the Content Framework to parse the content and create/update/delete a catalog entry in the MDC using this content URI as the entry's product URI. This parameter is used when the `directive` is set to `PROCESS`.

Update and Delete Request HTTP Header Parameters

Two optional HTTP header parameters are available on the update and delete RESTful URLs.

The directive header parameter allows the client to optionally direct the [Content Framework](#) to:

- only store the content in the DDF Content Repository (directive=STORE)
- store the content in the repository and parse the content to create a Metacard (directive=STORE_AND_PROCESS); this metacard is then created/updated/deleted in the Metadata Catalog (by invoking the [Catalog Framework](#) operations).

STORE_AND_PROCESS is the default value for the directive header parameter. The directive header parameter is only used on the PUT and DELETE RESTful URLs that have a contentId in the URL.

The contentUri header parameter allows the client to specify the URI of a product stored remotely/externally (relative to DDF). The contentUri header parameter is only used with the PUT and DELETE RESTful URLs where the contentId is *not* appended to the URL.

Sample Requests and Responses

The table below illustrates sample REST requests and their responses for each of the operations supported by the Content REST Endpoint.

For the examples below, DDF was running on host DDF_HOST on port DDF_PORT. Also, for all examples below the binary data, i.e., the "content", is not included in the request's body.

Operation	Request	Response
Create Content and Catalog Entry	<pre>POST http://DDF_HOST:DDF_PORT/services/content/ HTTP/1.1 Content-Type: multipart/form-data; boundary=ARCFomBoundaryuxprlpjxmakbj4i --ARCFomBoundaryuxprlpjxmakbj4i Content-Disposition: form-data; name="directive" STORE_AND_PROCESS --ARCFomBoundaryuxprlpjxmakbj4i Content-Disposition: form-data; name="file"; filename="C:\DDF\geojson_valid.json" Content-Type: application/json;id=geojson <content included in payload but omitted here for brevity> --ARCFomBoundaryuxprlpjxmakbj4i--</pre>	<pre>HTTP/1.1 201 Created Catalog-ID: e82a31253e634a409c83d7164638f029 Content-ID: ef0ef614bbdb4ede99e2371ebd2280ee Content-Length: 0 Content-URI: content:ef0ef614bbdb4ede99e2371ebd2280ee Date: Wed, 13 Feb 2013 21:56:15 GMT Location: http://127.0.0.1:8181/services/content/ef0ef614bbdb4ede99e2371ebd2280ee Server: Jetty(7.5.4.v20111024)</pre>
Create Content Only	<pre>POST http://DDF_HOST:DDF_PORT/services/content/ HTTP/1.1 Content-Type: multipart/form-data; boundary=ARCFomBoundaryuxprlpjxmakbj4i --ARCFomBoundaryuxprlpjxmakbj4i Content-Disposition: form-data; name="directive" STORE --ARCFomBoundaryuxprlpjxmakbj4i Content-Disposition: form-data; name="file"; filename="C:\DDF\geojson_valid.json" Content-Type: application/json;id=geojson <content included in payload but omitted here for brevity> --ARCFomBoundaryuxprlpjxmakbj4i--</pre>	<pre>HTTP/1.1 201 Created Content-ID: 7d671cd8e9aa4637960b37c7b3870aed Content-Length: 0 Content-URI: content:7d671cd8e9aa4637960b37c7b3870aed Date: Wed, 13 Feb 2013 21:56:16 GMT Location: http://127.0.0.1:8181/services/content/7d671cd8e9aa4637960b37c7b3870aed Server: Jetty(7.5.4.v20111024)</pre>

Create Catalog Entry Only	<p>POST http://DDF_HOST:DDF_PORT/services/content/ HTTP/1.1</p> <p>Content-Type: multipart/form-data; boundary=ARCFomBoundaryuxprlpjxmakbj4i</p> <p>--ARCFomBoundaryuxprlpjxmakbj4i Content-Disposition: form-data; name="directive"</p> <p>PROCESS</p> <p>--ARCFomBoundaryuxprlpjxmakbj4i Content-Disposition: form-data; name="contentUri"</p> <p>http://localhost:8080/some/path/file.json</p> <p>--ARCFomBoundaryuxprlpjxmakbj4i Content-Disposition: form-data; name="file"; filename="C:\DDF\geojson_valid.json" Content-Type: application/json;id=geojson</p> <p><i><content included in payload but omitted here for brevity></i></p> <p>--ARCFomBoundaryuxprlpjxmakbj4i--</p>	<p>HTTP/1.1 200 OK</p> <p>Catalog-ID: 94d8fae228a84e29a7396196542e2608</p> <p>Content-Length: 0</p> <p>Date: Wed, 13 Feb 2013 21:56:16 GMT</p> <p>Server: Jetty(7.5.4.v20111024)</p>
Update Content and Catalog Entry	<p>PUT http://DDF_HOST:DDF_PORT/services/content/bf9763c2e74d46f68a9ed591c4b74591 HTTP/1.1</p> <p>Accept-Encoding: gzip,deflate directive: STORE_AND_PROCESS Content-Type: application/json;id=geojson User-Agent: Jakarta Commons-HttpClient/3.1 Host: 127.0.0.1:8181 Content-Length: 9608</p> <p><i><content included in payload but omitted here for brevity></i></p>	<p>HTTP/1.1 200 OK</p> <p>Catalog-ID: d9ccbc9d139a4abbb0b1cdded1de0921</p> <p>Content-ID: bf9763c2e74d46f68a9ed591c4b74591</p> <p>Content-Length: 0</p> <p>Date: Wed, 13 Feb 2013 21:56:25 GMT</p> <p>Server: Jetty(7.5.4.v20111024)</p>
Update Content Only	<p>PUT http://DDF_HOST:DDF_PORT/services/content/bf9763c2e74d46f68a9ed591c4b74591 HTTP/1.1</p> <p>Accept-Encoding: gzip,deflate directive: STORE Content-Type: application/json;id=geojson User-Agent: Jakarta Commons-HttpClient/3.1 Host: 127.0.0.1:8181 Content-Length: 9608</p> <p><i><content included in payload but omitted here for brevity></i></p>	<p>HTTP/1.1 200 OK</p> <p>Content-ID: 7a702cd5c95347d2aa79ccc25b39e4f6</p> <p>Content-Length: 0</p> <p>Date: Wed, 13 Feb 2013 21:56:25 GMT</p> <p>Server: Jetty(7.5.4.v20111024)</p>

Update Catalog Entry Only	PUT http://DDF_HOST:DDF_PORT/services/content/bf9763c2e74d46f68a9ed591c4b74591 HTTP/1.1 Accept-Encoding: gzip,deflate directive: PROCESS Content-Type: application/json;id=geojson User-Agent: Jakarta Commons-HttpClient/3.1 Host: 127.0.0.1:8181 Content-Length: 9608 <content included in payload but omitted here for brevity>	HTTP/1.1 200 OK Catalog-ID: 65c36410e72b4fe295cc4d23da22d370 Content-Length: 0 Date: Wed, 13 Feb 2013 21:56:25 GMT Server: Jetty(7.5.4.v20111024)
Delete Content and Catalog Entry	DELETE http://DDF_HOST:DDF_PORT/services/content/911e27aba723448ea420142b0e793d38 HTTP/1.1 Accept-Encoding: gzip,deflate directive: STORE_AND_PROCESS User-Agent: Jakarta Commons-HttpClient/3.1 Host: 127.0.0.1:8181	HTTP/1.1 204 No Content Catalog-ID: 5236910acbd14d97a786f1fa95d43d58 Content-ID: 911e27aba723448ea420142b0e793d38 Content-Length: 0 Date: Wed, 13 Feb 2013 21:56:31 GMT Server: Jetty(7.5.4.v20111024)
Delete Content Only	DELETE http://DDF_HOST:DDF_PORT/services/content/eb91c8ee225d4cddb4d9fbe2d9bf5d7c HTTP/1.1 Accept-Encoding: gzip,deflate directive: STORE User-Agent: Jakarta Commons-HttpClient/3.1 Host: 127.0.0.1:8181	HTTP/1.1 204 No Content Content-ID: eb91c8ee225d4cddb4d9fbe2d9bf5d7c Content-Length: 0 Date: Wed, 13 Feb 2013 21:56:31 GMT Server: Jetty(7.5.4.v20111024)
Delete Catalog Entry Only	DELETE http://DDF_HOST:DDF_PORT/services/content/ HTTP/1.1 Accept-Encoding: gzip,deflate contentUri:http://DDF_HOST:DDF_PORT/some/path5/file.json User-Agent: Jakarta Commons-HttpClient/3.1 Host: 127.0.0.1:8181	HTTP/1.1 204 No Content Catalog-ID: c9a2b1c395f74300b33529483f095196 Content-Length: 0 Date: Wed, 13 Feb 2013 21:56:31 GMT Server: Jetty(7.5.4.v20111024)
Read	GET http://DDF_HOST:DDF_PORT/services/content/d34fd2b31f314aa6ade162015ba3016f HTTP/1.1 Accept-Encoding: gzip,deflate User-Agent: Jakarta Commons-HttpClient/3.1 Host: 127.0.0.1:8181	HTTP/1.1 200 OK Content-Length: 9579 Content-Type: application/json;id=geojson Date: Wed, 13 Feb 2013 21:56:24 GMT Server: Jetty(7.5.4.v20111024) ... (remaining data of content item retrieved omitted for brevity) ...

cURL Commands

The table below illustrates sample cURL commands corresponding to a few of the above REST requests. Pay special attention to the flags, as they vary between operations.

For these examples, DDF was running on host DDF_HOST on port DDF_PORT. We ingested/updated a file named geojson_valid.json whose MIME type was application/json;id=geojson, and whose content ID ended up being CONTENT_ID.

To perform each operation without using the catalog, replace STORE_AND_PROCESS with STORE, and to manipulate the catalog entry only, replace it with PROCESS.

Operation	Command
Create Content and Catalog Entry	curl -i -X POST -F "directive=STORE_AND_PROCESS" -F "filename=geojson_valid.json" -F "file=@geojson_valid.json;type=application/json;id=geojson" http://DDF_HOST:DDF_PORT/services/content/

Update Content and Catalog Entry	<code>curl -i -X PUT -H "directive: STORE_AND_PROCESS" -H "Content-Type: application/json;id=geojson" --data-binary "@geojson_valid.json" http://DDF_HOST:DDF_PORT/services/content/CONTENT_ID</code>
Delete Content and Catalog Entry	<code>curl -i -X DELETE -H "directive: STORE_AND_PROCESS" http://DDF_HOST:DDF_PORT/services/content/CONTENT_ID</code>
Read	<code>curl -i -X GET http://DDF_HOST:DDF_PORT/services/content/CONTENT_ID</code>

Installation and Uninstallation

The Content REST CRUD Endpoint, packaged as the `content-rest-endpoint` feature, can be installed and uninstalled using the normal processes described in the [Configuration](#) section.

Configuration

The Content REST CRUD Endpoint has no configurable properties. It can only be installed or uninstalled.

Known Issues

DDF Metrics Applications

Metrics Collecting Application

- [Description](#)
 - [Source Metrics](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Known Issues](#)

Description

The Metrics Collecting Application collects data for all of the pre-configured metrics in DDF and stores them in custom JMX Management Bean (MBean) attributes. Samples of each metric's data is collected every 60 seconds and stored in the `<DDF_INSTALL_DIR>/data/metrics` directory, with each metric stored in its own `.rrd` file. Refer to the [Metrics Reporting Application](#) for how the stored metrics data can be viewed.

Do not remove the `<DDF_INSTALL_DIR>/data/metrics` directory or any files in it. If this is done, then all existing metrics data will be permanently lost.

Also note that if DDF is uninstalled/re-installed that all existing metrics data will be permanently lost.

The metrics currently being collected by DDF are:

Metric	JMX MBean Name	MBean Attribute Name	Description
Catalog Exceptions	<code>ddf.metrics.catalog:name=Exceptions</code>	Count	A count of the total number of exceptions, of all types, thrown across all catalog queries executed.
Catalog Exceptions Federation	<code>ddf.metrics.catalog:name=Exceptions.Federation</code>	Count	A count of the total number of Federation exceptions thrown across all catalog queries executed.
Catalog Exceptions Source Unavailable	<code>ddf.metrics.catalog:name=Exceptions.SourceUnavailable</code>	Count	A count of the total number of SourceUnavailable exceptions thrown across all catalog queries executed. These exceptions occur when the source being queried is currently not available.

Catalog Exceptions Unsupported Query	ddf.metrics.catalog:name=Exceptions.UnsupportedQuery	Count	A count of the total number of UnsupportedQuery exceptions thrown across all catalog queries executed . These exceptions occur when the query being executed is not supported or is invalid.
Catalog Ingest Created	ddf.metrics.catalog:name=Ingest.Created	Count	A count of the number of catalog entries created in the Metadata Catalog.
Catalog Ingest Deleted	ddf.metrics.catalog:name=Ingest.Deleted	Count	A count of the number of catalog entries updated in the Metadata Catalog.
Catalog Ingest Updated	ddf.metrics.catalog:name=Ingest.Updated	Count	A count of the number of catalog entries deleted from the Metadata Catalog.
Catalog Queries	ddf.metrics.catalog:name=Queries	Count	A count of the number of queries attempted.
Catalog Queries Comparison	ddf.metrics.catalog:name=Queries.Comparison	Count	A count of the number of queries attempted that included a string comparison criteria as part of the search criteria , e.g., PropertyIsLike, PropertyIsEqualTo, etc.
Catalog Queries Federated	ddf.metrics.catalog:name=Queries.Federated	Count	A count of the number of federated queries attempted.
Catalog Queries Fuzzy	ddf.metrics.catalog:name=Queries.Fuzzy	Count	A count of the number of queries attempted that included a string comparison criteria with fuzzy searching enabled as part of the search criteria .
Catalog Queries Spatial	ddf.metrics.catalog:name=Queries.Spatial	Count	A count of the number of queries attempted that included a spatial criteria as part of the search criteria .
Catalog Queries Temporal	ddf.metrics.catalog:name=Queries.Temporal	Count	A count of the number of queries attempted that included a temporal criteria as part of the search criteria .
Catalog Queries Total Results	ddf.metrics.catalog:name=Queries.TotalResults	Mean	An average of the total number of results returned from executed queries . This total results data is averaged over the metric's sample rate.
Catalog Queries Xpath	ddf.metrics.catalog:name=Queries.Xpath	Count	A count of the number of queries attempted that included a Xpath criteria as part of the search criteria .
Catalog Resource Retrieval	ddf.metrics.catalog:name=Resource	Count	A count of the number of products retrieved.
Services Latency	ddf.metrics.services:name=Latency	Mean	The response time from receipt of the request at the endpoint until the response is about to be sent to the client from the endpoint . This response time data is averaged over the metric's sample rate.

Source Metrics

Metrics are also collected on a per Source basis for each configured Federated Source and Catalog Provider. When the Source is configured, the metrics listed in the table below are automatically created. With each request that is either an enterprise query or a query that lists the source(s) to query, these metrics are collected. When the Source is deleted, the associated metrics' MBeans and Collectors are also deleted. However, the RRD file in the `data/metrics` directory containing the collected metrics remain indefinitely and remain accessible from the Metrics tab in the Admin console.

In the table below, the metric name is based on the Source's ID (indicated by `<sourceId>`).

Metric	JMX MBean Name	MBean Attribute Name	Description
--------	----------------	----------------------	-------------

Source <sourceId> Exceptions	ddf.metrics.catalog.source:name=<sourceId>.Exceptions	Count	A count of the total number of exceptions, of all types, thrown from catalog queries executed on this Source.
Source <sourceId> Queries	ddf.metrics.catalog.source:name=<sourceId>.Queries	Count	A count of the number of queries attempted on this Source.
Source <sourceId> Queries Total Results	ddf.metrics.catalog.source:name=<sourceId>.Queries.TotalResults	Mean	An average of the total number of results returned from executed queries on this Source. This total results data is averaged over the metric's sample rate.

Usage

The Metrics Collecting Application is used when collection of historical metrics data, such as catalog query metrics or message latency type of data, is desired.

Installation and Uninstallation

The Metrics Collecting Application is installed by default. It should not be uninstalled because the JMX Collector will not be removed. The next time the system is restarted, the JMX Collector will try to find the JMX MBean but it won't be created since the metrics bundle was uninstalled. It will timeout after a few minutes.

Configuration

No configuration is made for the Metrics Collecting Application. All of the metrics that it collects data on are either pre-configured in DDF out of the box, or dynamically created

as Sources are created or deleted.

Known Issues

- None

Metrics Reporting Application

- [Description](#)
- [Usage](#)
 - [Metric Data Supported Formats](#)
 - [Metrics Aggregate Reports](#)
 - [Adding Custom Metrics to the Metrics Tab](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Known Issues](#)

Description

The DDF Metrics Reporting Application provides access to historical data in graphical, comma-separated values file, Excel spreadsheet, PowerPoint, XML, and JSON formats for system metrics collected while DDF is running. Aggregate reports (weekly, monthly, and yearly) are also provided where **all** collected metrics are included in the report. Aggregate reports are available in Excel and PowerPoint formats.

Usage

The DDF Metrics Reporting Application provides a web console plugin that adds a new tab to the Admin console entitled "Metrics". When selected, this Metrics tab displays a list of all of the metrics being collected by DDF, e.g., Catalog Queries, Catalog Queries Federated, Catalog Ingest Created, etc.

With each metric in the list, a set of hyperlinks is displayed under each column entitled with the available time ranges. The time ranges currently supported, all measured from the current time that the hyperlink is selected, are: 15 minutes, 1 hour, 4 hours, 12 hours, 1 day, 3 days, 1 week, 1 month, 3 months, 6 months, and 1 year.

All metrics reports are generated by accessing the collected metric data stored in the <DDF_INSTALL_DIR>/data/metrics directory. All files in this directory are generated by the JmxCollector using RRD4J, a Round Robin Database for Java open source product. All files in this directory will have the .rrd file extension and are binary files, hence they cannot be opened directly. These files should only be accessed using the Metrics tab's hyperlinks. There is one RRD file per metric being collected. Each RRD file is sized at creation time and will never increase in size as data is collected. One year's worth of metric data requires approximately 1 MB file storage.

Do not remove the <DDF_INSTALL_DIR>/data/metrics directory or any files in it. If this is done, then all existing metrics data will

be permanently lost.

Also note that if DDF is uninstalled/re-installed that all existing metrics data will be permanently lost.

There is a hyperlink per format that the metric's historical data can be displayed in. For example, the PNG hyperlink for 15m for the Catalog Queries metric maps to `http://<DDF_HOST>:<DDF_PORT>/services/internal/metrics/catalogQueries.png?dateOffset=900` where the `dateOffset=900` indicates 900 seconds (15 minutes) from current time to graph data.

Metrics

Admin	Bundles	Configuration	Configuration Status	Deployment Packages	Events	Features	Gogo	Licenses	Log Service	Metrics	OSGI Repository	Services	Shell	System Information	
Metric			15m	1h		1d		1w		1M		3M		6M	1y
Services Latency			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Exceptions			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Exceptions Federation			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Exceptions Source Unavailable			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Exceptions Unsupported Query			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Ingest Created			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Ingest Deleted			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Ingest Updated			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Queries			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Queries Comparison			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Queries Federated			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Queries Fuzzy			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Queries Spatial			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Queries Temporal			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Queries Total Results			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Queries Xpath			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS
Catalog Resource Retrieval			PNG CSV XLS	PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS		PNG CSV XLS	PNG CSV XLS

Weekly Reports

April 15 - April 21	XLS	PPT
April 08 - April 14	XLS	PPT
April 01 - April 07	XLS	PPT
March 25 - March 31	XLS	PPT

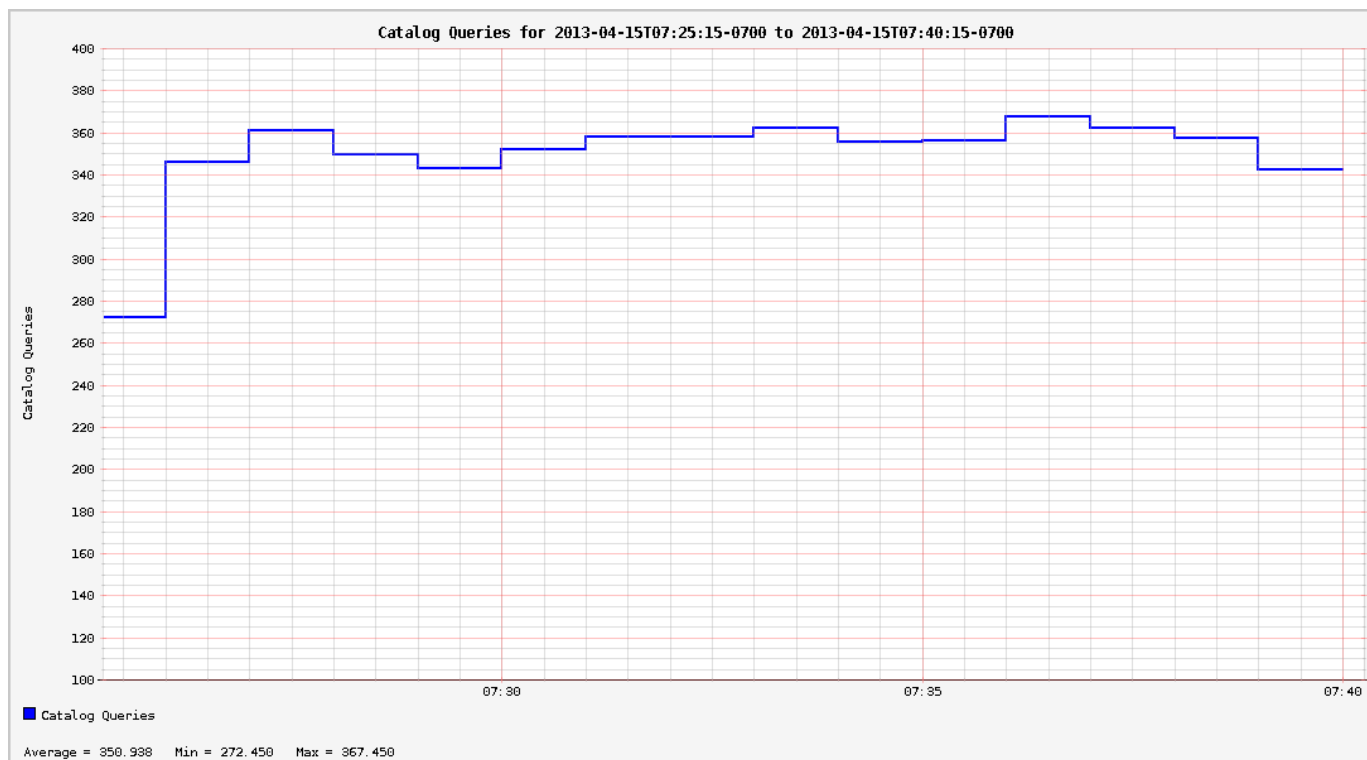
Monthly Reports

March, 2013	XLS	PPT
February, 2013	XLS	PPT
January, 2013	XLS	PPT
December, 2012	XLS	PPT
November, 2012	XLS	PPT
October, 2012	XLS	PPT
September, 2012	XLS	PPT
August, 2012	XLS	PPT
July, 2012	XLS	PPT
June, 2012	XLS	PPT
May, 2012	XLS	PPT
April, 2012	XLS	PPT

Yearly Reports

2012	XLS	PPT
------	-----	-----

All of the metric graphs displayed are in PNG format and are displayed on their own page. The user must hit the back button in the browser to return to the Admin console, or the user when selecting the hyperlink for a graph can use the right mouse button in the browser to display the graph in a separate browser tab or window, thereby keeping the Admin console also displayed. The screen shot below is a sample graph of the Catalog Queries metric's data for the previous 15 minutes from when the link was selected. Note that the y-axis label and the title use the metric's name (Catalog Queries) by default. The average, min, and max of all of the metric's data is summarized in the lower left of the graph.



The user can also specify custom time ranges by adjusting the URL used to access the metric's graph. The Catalog Queries metric data could be graphed for a specific time range by specifying the `startDate` and `endDate` query parameters in the URL.

Note that the Metrics Endpoint URL has "internal" in it. This indicates that this endpoint is intended for internal use by the DDF code. As such this endpoint is likely to change in future versions, hence any custom applications built to make use of it, as described below, should be done with caution.

For example, to map the Catalog Queries metric data for March 31, 6:00 am to April 1, 2013, 11:00 am (Arizona timezone, which is -07:00) the URL would be:

```
http://<DDF_HOST><DDF_PORT>/services/internal/metrics/catalogQueries.png?startDate=2013-03-31T06:00:00-07:00&endDate=2013-04-01T11:00:00-07:00
```

Or to view the last 30 minutes of data for the Catalog Queries metric a custom URL with a `dateOffset=1800` (30 minutes in seconds) could be used:

```
http://<DDF_HOST>:<DDF_PORT>/services/internal/metrics/catalogQueries.png?dateOffset=1800
```

The table below lists all of the options for the Metrics Endpoint URL to execute custom metrics data requests:

Parameter	Description	Example
startDate	Specifies the start of the time range of the search on the metric's data (RFC-3339 - Date and Time format, i.e. YYYY-MM-DDTHH:mm:ssZ). Date/time must be earlier than the endDate. <i>This parameter cannot be used with the dateOffset parameter.</i>	startDate=2013-03-31T06:00:00-07:00
endDate	Specifies the end of the time range of the search on the metric's data (RFC-3339 - Date and Time format, i.e. YYYY-MM-DDTHH:mm:ssZ). Date/time must be later than the startDate. <i>This parameter cannot be used with the dateOffset parameter.</i>	endDate=2013-04-01T11:00:00-07:00

dateOffset	Specifies an offset, backwards from the current time, to search on the modified time field for entries. Defined in seconds and must be a positive Integer. <i>This parameter cannot be used with the startDate or endDate parameters.</i>	dateOffset=1800
yAxisLabel	(optional) the label to apply to the graph's y-axis. Will default to the metric's name, e.g., Catalog Queries. <i>This parameter is only applicable for the metric's graph display format.</i>	Catalog Query Count
title	(optional) the title to be applied to the graph. Will default to the metric's name plus the time range used for the graph. <i>This parameter is only applicable for the metric's graph display format.</i>	Catalog Query Count for the last 15 minutes

Metric Data Supported Formats

The metric's historical data can be displayed in several formats, including the PNG format previously discussed, CSV file, Excel .xls file, PowerPoint .ppt file, XML file, and JSON file. The PNG, CSV, and XLS formats are accessed via hyperlinks provided in the Metrics tab web page. The PPT, XML, and JSON formats are accessed only by specifying the format in the custom URL, e.g., http://<DDF_HOST>:<DDF_PORT>/services/internal/metrics/catalogQueries.json?dateOffset=1800

The table below describes each of the supported formats, how to access them, and an example where applicable. (NOTE: all example URLs begin with http://<DDF_HOST>:<DDF_PORT> which is omitted in the table for brevity).

Display Format	Description	How To Access	Example URL
PNG	Displays the metric's data as a PNG-formatted graph where the x-axis is time and the y-axis is the metric's sampled data values.	Via hyperlink on the Metrics tab or directly via custom URL.	Accessing Catalog Queries metric data for last 8 hours (8 * 60 * 60 = 28800 seconds): /services/internal/metrics/catalogQueries.png?dateOffset=28800&yAxisLabel=my%20label&title=my%20graph%20title Accessing Catalog Queries metric data between 6:00 am March 10, 2013 and 10:00 am April 2, 2013: /services/internal/metrics/catalogQueries.png?startDate=2013-03-10T06:00:00-07:00&endDate=2013-04-02T10:00:00-07:00&yAxisLabel=my%20label&title=my%20graph%20title <i>Note that the yAxisLabel and title parameters are optional.</i>
CSV	Displays the metric's data as a Comma-Separated Value (CSV) file, which can be auto-displayed in Excel based on browser settings. The generated CSV file will consist of 2 columns of data, Timestamp and Value, where the first row are the column headers and the remaining rows are the metric's sampled data over the specified time range.	Via hyperlink on the Metrics tab or directly via custom URL.	Accessing Catalog Queries metric data for last 8 hours (8 * 60 * 60 = 28800 seconds): /services/internal/metrics/catalogQueries.csv?dateOffset=28800 Accessing Catalog Queries metric data between 6:00 am March 10, 2013 and 10:00 am April 2, 2013: /services/internal/metrics/catalogQueries.csv?startDate=2013-03-10T06:00:00-07:00&endDate=2013-04-02T10:00:00-07:00

XLS	<p>Displays the metric's data as an Excel (XLS) file, which can be auto-displayed in Excel based on browser settings.</p> <p>The generated XLS file will consist of:</p> <ul style="list-style-type: none"> • title in first row based on metric's name and specified time range • Column Headers for Timestamp and Value • 2 columns of data containing the metric's sampled data over the specified time range • the total count, if applicable, in the last row 	<p>Via hyperlink on the Metrics tab</p> <p>or directly via custom URL.</p>	<p>Accessing Catalog Queries metric data for last 8 hours (8 * 60 * 60 = 28800 seconds):</p> <p><code>/services/internal/metrics/catalogQueries.xls?dateOffset=28800</code></p> <p>Accessing Catalog Queries metric data between 6:00 am March 10, 2013 and 10:00 am April 2, 2013:</p> <p><code>/services/internal/metrics/catalogQueries.xls?startDate=2013-03-10T06:00:00-07:00&endDate=2013-04-02T10:00:00-07:00</code></p>
PPT	<p>Displays the metric's data as a PowerPoint (PPT) file, which can be auto-displayed in PowerPoint based on browser settings.</p> <p>The generated PPT file will consist of a single slide containing:</p> <ul style="list-style-type: none"> • a title based on the metric's name • the metric's PNG graph embedded as a picture in the slide • the total count, if applicable 	<p>via custom URL only</p>	<p>Accessing Catalog Queries metric data for last 8 hours (8 * 60 * 60 = 28800 seconds):</p> <p><code>/services/internal/metrics/catalogQueries.ppt?dateOffset=28800</code></p> <p>Accessing Catalog Queries metric data between 6:00 am March 10, 2013 and 10:00 am April 2, 2013:</p> <p><code>/services/internal/metrics/catalogQueries.ppt?startDate=2013-03-10T06:00:00-07:00&endDate=2013-04-02T10:00:00-07:00</code></p>

XML	Displays the metric's data as an XML-formatted file	via custom URL only	<p>Accessing Catalog Queries metric data for last 8 hours (8 * 60 * 60 = 28800 seconds):</p> <pre>/services/internal/metrics/catalogQueries.xml?dateOffset=28800</pre> <p>Accessing Catalog Queries metric data between 6:00 am March 10, 2013 and 10:00 am April 2, 2013:</p> <pre>/services/internal/metrics/catalogQueries.xml? startDate=2013-03-10T06:00:00-07:00&endDate=2013-04-02T10:00:00-07:00</pre> <p>Sample XML-formatted output would look like:</p> <div><pre><catalogQueries> <title>Catalog Queries for Apr 15 2013 08:45:53 to Apr 15 2013 09:00:53</title> <data> <sample> <timestamp>Apr 15 2013 08:45:00</timestamp> <value>361</value> </sample> <sample> <timestamp>Apr 15 2013 09:00:00</timestamp> <value>353</value> </sample> <totalCount>5721</totalCount> </data> </catalogQueries></pre></div>
-----	---	---------------------	--

JSON	Displays the metric's data as an JSON-formatted file	via custom URL only	<p>Accessing Catalog Queries metric data for last 8 hours (8 * 60 * 60 = 28800 seconds):</p> <pre>/services/internal/metrics/catalogQueries.json?dateOffset=28800</pre> <p>Accessing Catalog Queries metric data between 6:00 am March 10, 2013 and 10:00 am April 2, 2013:</p> <pre>/services/internal/metrics/catalogQueries.json? startDate=2013-03-10T06:00:00-07:00&endDate=2013-04-02T10:00:00-07:00</pre> <p>Sample JSON-formatted output would look like:</p> <pre>{ "title": "Query Count for Jul 9 1998 09:00:00 to Jul 9 1998 09:50:00", "totalCount": 322, "data": [{ "timestamp": "Jul 9 1998 09:20:00", "value": 54 }, { "timestamp": "Jul 9 1998 09:45:00", "value": 51 }] }</pre>
------	--	---------------------	---

Metrics Aggregate Reports

The Metrics tab also provides aggregate reports for the collected metrics. These are reports that include data for all of the collected metrics for the specified time range.

The aggregate reports provided are:

- Weekly reports - for each week up to the past 4 **complete** weeks from current time. A complete week is defined as a week from Monday through Sunday. For example, if current time is Thursday, April 11, 2013, then the past complete week would be from April 1 through April 7.
- Monthly reports - for each month up to the past 12 **complete** months from current time. A complete month is defined as the full month(s) preceding current time. For example, if current time is Thursday, April 11, 2013, then the past complete 12 months would be from April 2012 through March 2013.
- Yearly reports - for the past **complete** year from current time. A complete year is defined as the full year preceding current time. For example, if current time is Thursday, April 11, 2013, then the past complete year would be 2012.

An aggregate report in XLS format would consist of a single workbook (spreadsheet) with multiple worksheets in it, where a separate worksheet exists for each collected metric's data. Each worksheet would display:

- the metric's name and the time range of the collected data,
- 2 columns, Timestamp and Value, for each sample of the metric's data that was collected during the time range
- a total count (if applicable) at the bottom of the worksheet

An aggregate report in PPT format would consist of a single slideshow with a separate slide for each collected metric's data. Each slide would display:

- a title with the metric's name
- the PNG graph for the metric's collected data during the time range
- a total count (if applicable) at the bottom of the slide

Hyperlinks are provided for each aggregate report's time range in the supported display formats, which include Excel (XLS) and PowerPoint (PPT). Aggregate reports for custom time ranges can also be accessed directly via the URL `http://<DDF_HOST>:<DDF_PORT>/services/internal/metrics/report.<format>?startDate=<start_date_value>&endDate=<end_date_value>` where `<format>` is either `xls` or `ppt` and the `<start_date_value>` and `<end_date_value>` specify the custom time range for the report.

The table below list several examples for custom aggregate reports. (NOTE: all example URLs begin with `http://<DDF_HOST>:<DDF_PORT>` which is omitted in the table for brevity).

Description	URL
XLS aggregate report for March 15, 2013 to April 15, 2013	/services/internal/metrics/report.xls?startDate=2013-03-15T12:00:00-07:00&endDate=2013-04-15T12:00:00-07:00
XLS aggregate report for last 8 hours	/services/internal/metrics/report.xls?dateOffset=28800
PPT aggregate report for March 15, 2013 to April 15, 2013	/services/internal/metrics/report.ppt?startDate=2013-03-15T12:00:00-07:00&endDate=2013-04-15T12:00:00-07:00
PPT aggregate report for last 8 hours	/services/internal/metrics/report.ppt?dateOffset=28800

Adding Custom Metrics to the Metrics Tab

It is possible to add custom (or existing, but non-collected) metrics to the Metrics Tab by writing an application. Refer to the SDK example source code for Sample Metrics located in the DDF source code at `sdk/sample-metrics` and `sdk/sdk-app`.

The Metrics framework is not an open API, but rather a closed, internal framework which can change at any time in future releases. Please be aware that any custom code written may not work with future releases.

Installation and Uninstallation

The Metrics Reporting Application can be installed and uninstalled using the normal processes described in the [Configuration](#) section.

Configuration

No configuration can be made for the Metrics Reporting Application. All of the metrics that it collects data on are pre-configured in DDF out of the box.

The `metrics-reporting` feature can only be installed and uninstalled. It is installed by default.

Known Issues

- The Metrics Collecting Application uses a “round robin” database, one that does not store individual values but, instead, stores the rate of change between values at different times. Due to the nature of this method of storage, along with the fact that some processes can cross time frames, small discrepancies - differences in values of one or two have been experienced - may appear in values for different time frames. These will be especially apparent for reports covering shorter time frames such as 15 minutes or 1 hour. These are due to the averaging of data over time periods and should not impact the values over longer periods of time.

DDF Mime Applications

DDF Mime Core

- [Mime Type Mapper](#)
- [DDF Mime Type Mapper](#)
- [Mime Type Resolver](#)
- [Custom Mime Type Resolver](#) — allows the `ApplicationName` to add support for custom mime types dynamically

Mime Type Mapper

Description

The `MimeTypeMapper` is the entry point in DDF for resolving file extensions to mime types, and vice versa.

`MimeTypeMappers` are used by the [ResourceReader](#) to determine the file extension for a given mime type in aid of retrieving a product. `MimeTypeMappers` are also used by the [FileSystemProvider](#) in the Content Framework to read a file from the content file repository.

The `MimeTypeMapper` maintains a list of all of the [MimeTypeResolvers](#) in DDF.

The `MimeTypeMapper` accesses each [MimeTypeResolver](#), according to its priority, until the provided file extension is successfully mapped to its corresponding mime type. If no mapping is found for the file extension, then `null` is returned for the mime type. Similarly, the `MimeTypeMapper` accesses each [MimeTypeResolver](#), according to its priority, until the provided mime type is successfully mapped to its corresponding file extension. If no mapping is found for the mime type, then `null` is returned for the file extension.

Examples

Examples of `MimeTypeMappers` provided in DDF are:

[DDF Mime TypeMapper](#)

DDF Mime Type Mapper

- [Description](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)

Description

The DDF Mime Type Mapper provides the reference implementation of a [MimeTypeMapper](#) that implements all requirements of the DDF Mime API. `MimeTypeMapperImpl` is the implementation of the DDF [MimeTypeMapper](#).

Usage

The DDF Mime Type Mapper is the core implementation of the DDF Mime API. It provides access to all [MimeTypeResolvers](#) within DDF, which provide mapping of mime types to file extensions, and file extensions to mime types.

Installation and Uninstallation

The DDF Mime Type Mapper is bundled in the `mime-core` feature, which is part of the `mime-core-app` application. This feature can be installed and uninstalled using the normal processes described in the [Configuration](#) section.

The `mime-core` feature is installed by default.

Configuration

There is no configuration for this feature.

Mime Type Resolver

Description

A `MimeTypeResolver` is a DDF service that can map a file extension to its corresponding mime type, and conversely, can map a mime type to its file extension.

`MimeTypeResolvers` are assigned a priority (0-100, with the higher the number indicating the higher priority). This priority is used to sort all of the `MimeTypeResolvers` in the order they should be checked for mapping a file extension to a mime type (or vice versa). This priority also allows custom `MimeTypeResolvers` to be invoked before default `MimeTypeResolvers` if the custom resolver's priority is set higher than the default's.

`MimeTypeResolvers` are not typically invoked directly. Rather, the [MimeTypeMapper](#) maintains a list of `MimeTypeResolvers` (sorted by their priority) that it invokes to resolve a mime type to its file extension (or to resolve a file extension to its mime type).

Examples

Examples of `MimeTypeResolvers` in DDF include:

[Tika MimeTypeResolver](#)

[Custom MimeTypeResolver](#)

Custom Mime Type Resolver

- [Description](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
 - [Configuring Custom Mime Types](#)
 - [Managed Service Factory PID](#)
 - [Configurable Properties](#)
- [Implementation Details](#)
 - [Imported Services](#)
 - [Exported Services](#)

Description

The Custom Mime Type Resolver is a [MimeTypeResolver](#) that defines the custom mime types that DDF will support out of the box. These are mime types not supported by the default [TikaMimeTypeResolver](#).

Currently the custom mime types supported by the Custom Mime Type Resolver that are configured for DDF out-of-the-box are:

File Extension	Mime Type
nitf	image/nitf
ntf	image/nitf
xml	text/xml;id=ddms
json	json=application/json;id=geojson

New custom mime type resolver mappings can be added using the Admin console.

As a [MimeTypeResolver](#), the Custom Mime Type Resolver will provide methods to map the file extension to the corresponding mime type, and vice versa.

Usage

The Custom Mime Type Resolver is used when mime types that are not supported by DDF out of the box need to be added. By adding custom mime type resolvers to DDF new content with that mime type can be processed by DDF.

Installation and Uninstallation

One Custom Mime Type Resolver is configured and installed out of the box for the image/nitf mime type. This custom resolver is bundled in the `mime-core-app` application and is part of the `mime-core` feature. This feature can be installed and uninstalled using the normal processes described in the [Configuration](#) section.

Additional Custom Mime Type Resolvers can be added for other custom mime types.

Configuration

This component can be configured using the normal processes described in the [Configuration](#) section.

The configurable properties for the Custom Mime Type Resolver are accessed from the *MIME Custom Types* Configuration in the Admin Console.

Configuring Custom Mime Types

Managed Service Factory PID

DDF_Custom_Mime_Type_Resolver
Configurable Properties

Title	Property	Type	Description	Default Value	Required
Resolver Name	name	String	Unique name for the custom mime type resolver	N/A	Yes
Priority	priority	Integer	Execution priority of the resolver. Range is 0 to 100, with 100 being the highest priority.	10	Yes
File Extensions to Mime Types	customMimeTypes	String	Comma-delimited list of key/value pairs where key is the file extension and value is the mime type, e.g., nitf=image/nitf	N/A	Yes

Implementation Details

Imported Services

Registered Interface	Availability	Multiple
<code>ddf.catalog.transform.InputTransformer</code>	optional	true

<code>ddf.catalog.transform.QueryResponseTransformer</code>	optional	true
<code>ddf.mime.MimeTypeResolver</code>	optional	true

Exported Services

Registered Interface	Service Property	Value
<code>ddf.mime.MimeTypeToTransformerMapper</code>		
<code>ddf.mime.MimeTypeMapper</code>		

DDF Mime Tika

- [Tika Mime Type Resolver](#) — allows ApplicationName to support numerous default mime types

Tika Mime Type Resolver

- [Description](#)
- [Usage](#)
- [Installation and Uninstallation](#)
- [Configuration](#)
- [Implementation Details](#)
 - [Exported Services](#)

Description

The `TikaMimeTypeResolver` is a [MimeTypeResolver](#) that is implemented using the [Apache Tika](#) open source product.

Using the Apache Tika content analysis toolkit, the `TikaMimeTypeResolver` provides support for resolving over 1300 mime types. (The `tika-mimetypes.xml` file that Apache Tika uses to define all of its default mime types that it supports is attached to this page.)

The `TikaMimeTypeResolver` is assigned a default priority of -1 to insure that it is always invoked last by the [MimeTypeMapper](#). This insures that any custom [MimeTypeResolvers](#) that may be installed will be invoked before the `TikaMimeTypeResolver`.

Usage

The `TikaMimeTypeResolver` provides the bulk of the default mime type support for DDF.

Installation and Uninstallation

The `TikaMimeTypeResolver` is bundled as the `mime-tika-resolver` feature in the `mime-tika-app` application. This feature can be installed and uninstalled using the normal processes described in the [Configuration](#) section.

This feature is installed by default.

Configuration

There are no configuration properties for the `mime-tika-resolver`.

Implementation Details

Exported Services

Registered Interface	Service Property	Value
<code>ddf.mime.MimeTypeResolver</code>		

DDF Platform Application

Description

The Platform Application is considered to be a core application of the distribution. The Platform Application has fundamental building blocks that the distribution needs in order to run. These building blocks include subsets of [Karaf](#), [CXF](#), [Cellar](#), and [Camel](#). Included as part of the Platform Application is also a Command Scheduler. The Command Scheduler allows users to schedule [Command Line Shell Commands](#) to run at certain specified intervals.

Usage

The Platform Application is a core building block for any application and should be referenced for its core component versions so that developers can ensure compatibility with their own applications. The Command Scheduler that is included in the Platform Application should be used by those that need or like the convenience of a "platform independent" method of running certain commands such as backing up data or logging settings. More information can be found on the [Command Scheduler](#) page.

Installation and Uninstallation

This application comes out of the box and is not intended to be uninstalled. The Command Scheduler is found within the `platform-scheduler` feature and can be installed and uninstalled using the normal processes described in the [Configuration](#) section.

Configuration

This Component can be configured using the normal processes described in the [Configuration](#) section. The configurable properties are accessed from the *Schedule Command* Configuration in the Admin Console.

Configurable Properties

Property	Type	Description	Default Value	Required
command	String	Shell command to be used within the container. For example, <code>log:set DEBUG</code>		yes
intervalInSeconds	Integer	The interval of time in seconds between each execution. Must be a positive integer. For example, 3600 is 1 hour.		yes

DDF Security Applications

Description

The included Security Applications provide Authentication, Authorization, and Auditing services for the DDF. They comprise both a framework that developers and integrators can extend and also a reference implementation that can be used which meets security requirements. More information about the security framework and how everything works as a single security solution can be found on the [Web Service Security](#) page.

The following pages give a description of the various applications, their containing bundles, configuration settings for the bundles, and which services are exported and imported.

Installation and Uninstallation

The Security Applications can be installed and uninstalled using the normal processes described in the [Configuration](#) section. Within the pages for each of the applications are specific installation instructions for the bundles.

Configuration

This Component can be configured using the normal processes described in the [Configuration](#) section. Within the pages for each of the applications are specific instructions on the configurations for the bundles and any additional information that may help decide how the configuration should be set for use cases.

Applications

- [Security Core](#)
- [Security CAS](#)
- [Security Encryption](#)
- [Security PEP](#)
- [Security PDP](#)
- [Security STS](#)

Security CAS

Description

The Security CAS app contains all of the services and implementations needed to integrate with the Central Authentication Server (CAS). Information on setting up and configuring the CAS server is located on the [CAS SSO Configuration](#) page.

Components

Bundle Name	Feature Located In	Description / Link to Bundle Page
security-cas-client	security-cas-client	Security CAS Client
security-cas-impl	security-cas-client	Security CAS Implementation
security-cas-tokenvalidator	security-cas-tokenvalidator	Security CAS Token Validator
security-cas-cxfServletfilter	security-cas-cxfServletfilter	Security CAS CXF Servlet Filter
security-cas-server		Security CAS Server

Security CAS Client

Description

The Security CAS Client bundle contains client files needed by components that are performing authentication with CAS. This includes setting up the CAS SSO servlet filters and also starting a callback service that is needed to request proxy tickets from CAS.

Configuration

Installation

This bundle is not installed by default and can be added by installing the `security-cas-client` feature.

Settings

Configuration Name	Default Value	Additional Description
Server Name	https://server:8993	This is the name of the server that is calling CAS. The URL is used during CAS redirection to redirect back to the calling server.
CAS Server URL	https://cas:8443/cas	The main URL to the CAS Web Application
CAS Server URL Prefix	/login	The suffix to the cas login url.
Proxy Callback URL	https://server:8993/sso	Full URL of the callback service that CAS hits to create proxy tickets.
Proxy Receptor URL	/sso	

Implementation Details

Imported Services

None.

Exported Services

Registered Interface	Implementation Class	Properties Set
javax.servlet.Filter	ddf.security.cas.client.ProxyFilter	CAS Filters

Security CAS Implementation

Description

The Security CAS Implementation bundle contains CAS-specific implementations of classes from the Security Core API. Inside of this bundle is the `ddf.security.service.impl.cas.CasAuthenticationToken` class. It is an implementation of the `AuthenticationToken` class that is used to pass Authentication Credentials to the Security Framework.

Configuration

None.

Implementation Details

Imported Services

None.

Exported Services

None.

Security CAS Server

Description

The Security CAS Server project creates a web application (war) file that is configured to be deployed to a tomcat application server. Information on installing and configuring it within tomcat is available on the [CAS SSO Configuration](#) page.

Configuration

N/A - Not a bundle.

Implementation Details

N/A - Not a bundle.

Security CAS Token Validator

Description

The Security CAS TokenValidator bundle exports a TokenValidator service that is called by the STS to validate CAS proxy tickets.

Configuration

Installation

This bundle is not installed by default and can be added by installing the `security-cas-tokenvalidator` feature.

Settings

Configuration Name	Default Value	Additional Description
CAS Server URL	https://localhost:8443/cas/	The hostname in the URL should match the hostname alias defined within the certificate that CAS is using for SSL communication.

Implementation Details

Imported Services

Registered Interface	Availability	Multiple
<code>ddf.security.encryption.EncryptionService</code>	required	false

Exported Services

Registered Interfaces	Implementation Class	Properties Set
-----------------------	----------------------	----------------

ddf.catalog.util.DdfConfigurationWatcher org.apache.cxf.sts.token.validator.TokenValidator	ddf.security.cas.WebSSOTokenValidator	CAS Server URL and Encryption Service reference
---	---------------------------------------	---

Security CAS CXF Servlet Filter

Description

The Security CAS CXF Servlet Filter bundle binds a list of CAS servlet filters to the CXF servlet. The servlet filters are defined by the security-cas-client bundle.

Configuration

Installation

This bundle is not installed by default and can be added by installing the `security-cas-cxf-servlet-filter` feature.

Settings

Configuration Name	Default Value	Additional Description
URL Pattern	/services/catalog/*	This defines what servlet URL the CAS filter should bind to. By default they will bind to the REST and OpenSearch endpoints. The REST endpoint is called by the SearchUI when accessing individual metadata about a metacard and when accessing the metacard's thumbnail. An example of just securing the OpenSearch endpoint would be the value: /services/catalog/query

Endpoints that are secured by the CXF Servlet Filters will not currently work with federation. With the default settings, REST and OpenSearch federation TO the site with this feature installed will not work. Federation FROM this site, however, will work normally.

Implementation Details

Imported Services

Registered Interface	Availability	Multiple
javax.servlet.Filter	required	false

Exported Services

None (filter is exported inside of code and not via configuration)

Security Core

Description

The Security Core App contains all of the necessary components that are used to perform security operations (Authentication, Authorization, and Auditing) required in the framework.

Components

Bundle Name	Located in Feature	Description / Link to Bundle Page
security-core-api	security-core	Security Core API
security-core-impl	security-core	Security Core Implementation

security-core-commons	security-core	Security Core Commons
security-core-opensaml	security-core	This is a fragment bundle that adds the xacml2 profile to the default opensaml bundle that comes with the ddf platform.

Security Core API

Description

The Security Core API contains all of the DDF Security Framework APIs that are used to perform security operations within DDF. More information on the APIs can be found on the [Web Service Security](#) page.

Configuration

None.

Installation and Uninstallation

The Security Core App installs this bundle by default. Do not uninstall the Security Core API as it is integral to system function and is depended on by all of the other security services.

Implementation Details

Imported Services

None.

Exported Services

None.

Security Core Commons

Description

The Security Core Commons bundle contains helper and utility classes that are used within DDF to help with performing common security operations. Most notably, this bundle contains the `ddf.security.common.audit.SecurityLogger` class that performs the security audit logging within DDF.

Configuration

None.

Implementation Details

Imported Services

None.

Exported Services

None.

Security Core Implementation

Description

The Security Core Implementation contains the reference implementations for the [Security Core API](#) interfaces that come with the DDF distribution.

Configuration

None.

Installation and Uninstallation

The Security Core App installs this bundle by default. It is recommended to use this bundle as it contains the reference implementations for many classes used within the DDF Security Framework.

Implementation Details

Imported Services

Registered Interface	Availability	Multiple
org.apache.shiro.realm.Realm	optional	true

Exported Services

Registered Interface	Implementation Class	Properties Set
ddf.security.service.SecurityManager	ddf.security.service.impl.SecurityManagerImpl	None

Security Encryption

Description

The DDF Security Encryption application offers an encryption framework and service implementation for other applications to use. This service is commonly used to encrypt and decrypt default passwords that are located within the metatype and administration web console.

Components

Bundle Name	Feature Located In	Description / Link to Bundle Page
security-encryption-api	security-encryption	Security Encryption API
security-encryption-impl	security-encryption	Security Encryption Implementation
security-encryption-commands	security-encryption	Security Encryption Commands

Security Encryption API

Description

The Security Encryption API bundle provides the framework for the encryption service. Applications that use the encryption service should import this bundle and use the interfaces defined within it instead of calling an implementation directly.

Configuration

Installation

This bundle is installed by default as part of the `security-encryption` feature. Many applications that come with DDF depend on this bundle and it should not be uninstalled.

Settings

None.

Implementation Details

Imported Services

None.

Exported Services

None.

Security Encryption Commands

Description

The Security Encryption Commands bundle enhances the DDF system console by allowing administrators and integrators to encrypt and decrypt values directly from the console. More information and sample commands are available on the [Encryption Service](#) page.

Configuration

Installation

This bundle is installed by default by the `security-encryption` feature. This bundle is tied specifically to the DDF console and can be uninstalled without causing any issues to other applications. When uninstalled, administrators will not be able to encrypt and decrypt data from the console.

Settings

None.

Implementation Details

Imported Services

None.

Exported Services

None.

Security Encryption Implementation

Description

The Security Encryption Implementation bundle contains all of the service implementations for the Encryption Framework and it exports those implementations as services to the OSGi service registry.

Configuration

Installation

This bundle is installed by default as part of the `security-encryption` feature. Other projects are dependent on the services this bundle exports and it should not be uninstalled unless another security service implementation is being added.

Settings

None.

Implementation Details

Imported Services

None.

Exported Services

Registered Interface	Implementation Class	Properties Set
<code>ddf.security.encryption.EncryptionService</code>	<code>ddf.security.encryption.impl.EncryptionServiceImpl</code>	Key

Security PDP

Description

The DDF Security PDP application contains services which are able to perform authorization decisions based on configurations and policies. In the DDF Security Framework these components are called Realms and they implement the `org.apache.shiro.realm.Realm` and `org.apache.shiro.authz.Authorizer` interfaces. Although these components perform decisions on access control, enforcement of this decision is performed by components within the [Security PEP](#) application.

Components

Bundle Name	Located in Feature	Description / Link to Bundle Page
<code>security-pdp-xacmlrealm</code>	<code>security-pdp-xacml</code>	Security PDP XACML Realm

security-pdp-authzrealm	security-pdp-simple	Security PDP AuthZ Realm
-------------------------	---------------------	--

Security PDP AuthZ Realm

Description

The DDF Security PDP AuthZ Realm exposes a Realm service that makes decisions on authorization requests using the attributes stored within the Metacard to determine if access should be granted. Unlike the [Security PDP XACML Realm](#), this realm does not use XACML and does not delegate decisions to an external processing engine. Decisions are made based on "match-all" and "match-one" logic. The configuration below provides the mapping between user attributes and Metacard attributes - one map exists for each type of mapping (each map may contain multiple values).

Match-All Mapping: This mapping is used to guarantee that all values present in the specified Metacard attribute exist in the corresponding user attribute.

Match-One Mapping: This mapping is used to guarantee that at least one of the values present in the specified Metacard attribute exists in the corresponding user attribute.

Configuration

Installation

This bundle is not installed by default and can be added by installing the `security-pdp-java` feature.

Settings

Settings can be found in the webconsole under Configuration -> Security Simple AuthZ Realm.

Configuration Name	Default Value	Additional Description
Roles	admin	Add all the roles that allow access to restricted actions. Any user that has any one of these roles will be allowed access to restricted actions.
Open Action List		Add any actions that will not be restricted by role. Any action listed here will automatically be allowed to be performed by any user in any role.
Match-All Mappings		These map user attributes to metacard security attributes to be used in "Match All" checking. All the values in the metacard attribute must be present in the user attributes in order to "pass" and allow access. These attribute names are case-sensitive.
Match-One Mappings		These map user attributes to metacard security attributes to be used in "Match One" checking. At least one of the values from the metacard attribute must be present in the corresponding user attribute to "pass" and allow access. These attribute names are case-sensitive.

Implementation Details

Imported Services

None

Exported Services

Registered Interfaces	Implementation Class	Properties Set
org.apache.shiro.realm.Realm	ddf.security.pdp.realm.SimpleAuthzRealm	None
org.apache.shiro.authz.Authorizer		

Security PDP XACML Realm

Description

The DDF Security PDP XACML Realm exposes a Realm that creates a XACML request with the incoming authorization information and sends the request to a XACML processing engine. The engine that gets sent the request is not hardcoded and is retrieved at runtime by the OSGi service registry. This realm contains an embedded XACML processing engine that handles the requests and policies.

Configuration

Installation

This bundle is not installed by default and can be added by installing the `security-pdp-xacml` feature.

Settings

None

Implementation Details

Imported Services

None

Exported Services

Registered Interfaces	Implementation Class	Properties Set
org.apache.shiro.realm.Realm	ddf.security.pep.realm.XACMLRealm	None
org.apache.shiro.authz.Authorizer		

Security PEP

Description

The DDF Security PEP application contains bundles and services that enable service and metacard authorization. These two types of authorization can be installed separately and extended with custom services.

Components

Bundle Name	Located in Feature	Description / Link to Bundle Page
security-pep-interceptor	security-pep-serviceauthz	Security PEP Interceptor
security-pep-redaction	security-pep-redaction	Security PEP Redaction

Security PEP Interceptor

Description

The Security PEP Interceptor bundle contains the `ddf.security.pep.interceptor.PEPAuthorizingInterceptor` class. This class uses CXF to intercept incoming SOAP messages and enforces service authorization policies by sending the service request to the security framework.

Configuration

Installation

This bundle is not installed by default and can be added by installing the `security-pep-serviceauthz` feature.

To perform service authorization within a default install of DDF, this bundle MUST be installed.

Settings

None.

Implementation Details

Imported Services

None.

Exported Services

None.

Security PEP Redaction

Description

The Security PEP Redaction bundle contains a redaction plugin that is added as a post query plugin in the DDF query lifecycle. This plugin looks at the security attributes on the metacard and compares them to the security attributes on the user who made the query request. If they do not match, the plug in will, depending on the configuration, either filter the metacard out of the results or will redact certain parts of the metacard that the user does not have access to see.

Configuration

Installation

This bundle is not installed by default and can be added by installing the `security-pep-redaction` feature.

Configuration

None

Implementation Details

Imported Services

None

Exported Services

Registered Interface	Implementation Class	Properties Set
ddf.catalog.plugin.PostQueryPlugin	ddf.security.pep.redaction.plugin.RedactionPlugin	None

Security STS

Description

The Security STS application contains the bundles and services necessary to run and talk to a Security Token Service (STS). It builds off of the Apache CXF STS code and add components specific to DDF functionality.

Components

Bundle Name	Located in Feature	Description / Link to Bundle Page
security-sts-clientconfig	security-sts-realm	Security STS Client Config
security-sts-realm	security-sts-realm	Security STS Realm
security-sts-ldaplogin	security-sts-ldaplogin	Security STS LDAP Login
security-sts-ldapclaimshandler	security-sts-server	Security STS LDAP Claims Handler
security-sts-server	security-sts-server	Security STS Server
security-sts-samlvalidator	security-sts-server	Contains the default CXF SAML validator, exposes it as a service for the STS.
security-sts-x509validator	security-sts-server	Contains the default CXF x509 validator, exposes it as a service for the STS.

Security STS Client Config

Description

The DDF Security STS Client Config bundle keeps track and exposes configurations and settings for the CXF STS Client. This client can be used by other services to create their own STS Client. Once a service is registered as a watcher of the configuration, it will be updated whenever the settings change for the sts client.

Configuration

Installation

This bundle is not installed by default and can be added by installing the `security-sts-realm` feature.

Settings

Settings can be found in the webconsole under Configuration -> Security STS Client.

Configuration Name	Default Value	Additional Information
STS Endpoint Name	{http://docs.oasis-open.org/ws-sx/ws-trust/200512/}STS_Port	
STS Service Name	{http://docs.oasis-open.org/ws-sx/ws-trust/200512/}SecurityTokenService	
STS Address	https://server:8993/services/SecurityTokenService	The hostname of the remote server should match the certificate that the server is using.
Username		Can be left blank if client is using credentials other than UsernameToken.
Password		Can be left blank if client is using credentials other than UsernameToken.
Signature Username		Must be set. Sets the username to use for the signature, example: client
Signature Properties	etc/ws-security/client/signature.properties	
Encryption Username		Must be set. Sets the username to use for encryption, example: tokenissuer
Encryption Properties	etc/ws-security/client/encryption.properties	
STS Token Username		Must be set. Sets the username associated with the STS Token, example: client
STS Properties	etc/ws-security/client/signature.properties	
Claims	<List of Claims>	

Implementation Details

Imported Services

Registered Interface	Availability	Multiple
ddf.catalog.DdfConfigurationWatcher	required	true
org.osgi.service.cm.ConfigurationAdmin	required	false

Exported Services

None

Security STS LDAP Claims Handler

Description

The DDF Security STS LDAP Claims Handler bundle adds functionality to the STS server that allows it to retrieve claims from an LDAP server. Additionally, it also adds mappings for the LDAP attributes to the STS SAML claims.

Configuration

Installation

This bundle is not installed by default and can be added by installing the `security-sts-server` feature.

Settings

Settings can be found in the webconsole under Configuration -> Security STS LDAP and Roles Claims Handler.

Configuration Name	Default Value	Additional Information
LDAP URL	ldap://localhost:1389	
LDAP Bind User DN	cn=admin	
LDAP Bind User Password	secret	This password value is encrypted by default using the Security Encryption application
LDAP Username Attribute	uid	
LDAP Base User DN	ou=users,dc=example,dc=com	
LDAP Base Group DN	ou=groups,dc=example,dc=com	
User Attribute Map File	etc/ws-security/attributeMap.properties	Properties file that contains mappings from Claim=LDAP attribute.

Implementation Details

Imported Services

Registered Interface	Availability	Multiple
ddf.security.encryption.EncryptionService	optional	false

Exported Services

Registered Interface	Implementation Class	Properties Set
org.apache.cxf.sts.claims.ClaimsHandler	ddf.security.sts.claimsHandler.LdapClaimsHandler	Properties from the settings
org.apache.cxf.sts.claims.ClaimsHandler	ddf.security.sts.claimsHandler.RoleClaimsHandler	Properties from the settings

Security STS LDAP Login

Description

The DDF Security STS LDAP Login bundle enables functionality within the STS that allows it to use an LDAP to perform authentication when passed a UsernameToken in a RequestSecurityToken SOAP request.

Configuration

Installation

This bundle is not installed by default and can be added by installing the `security-sts-ldaplogin` feature.

Settings

Configuration settings can be found in the webconsole under Configuration -> Security STS LDAP Login.

Configuration Name	Default Value	Additional Information
LDAP URL	ldaps://localhost:1636	

LDAP Bind User DN	cn=admin	
LDAP Bind User Password	secret	This password value is encrypted by default using the Security Encryption application.
LDAP Username Attribute	uid	
LDAP Base User DN	ou=users,dc=example,dc=com	
LDAP Base Group DN	ou=groups,dc=example,dc=com	
SSL Keystore Alias	server	This alias is used when connecting to the LDAP using SSL (LDAPS).

Implementation Details

Imported Services

None

Exported Services

None

Security STS Realm

Description

The DDF Security STS Realm is an Realm tha performs authentication of a user by delegating the authentication request to an STS. This is different than the realms located within the [Security PDP](#) application as those ones only perform authorization and not authentication.

Configuration

Installation

This bundle is installed by default and should not be uninstalled unless the security framework is not being used.

Settings

None

Implementation Details

Imported Services

Registered Interface	Availability	Multiple
ddf.security.encryption.EncryptionService	opt	false

Exported Services

Registered Interfaces	Implementation Class	Properties Set
ddf.catalog.util.DdfConfigurationWatcher	ddf.security.realm.sts.StsRealm	None
org.apache.shiro.realm.Realm		

Security STS Server

Description

The DDF Security STS Server is a bundle that starts up an implementation of the CXF STS. The STS obtains many of its configurations (Claims Handlers, Token Validators..etc) from the OSGi service registry as those items are registered as services using the cxf interfaces. The various services that the STS Server imports are listed further down in the Implementation Details section of this page.

The WSDL for the STS is located at the security-sts-server/sr/main/resources/META-INF/sts/wsd/ ws-trust-1.4-service.wsdl within the source code.

Configuration

Installation

This bundle is not installed by default and can be added by installing the `security-sts-server` feature.

Settings

Configuration settings can be found in the webconsole under Configuration -> Security STS Server.

Configuration Name	Default Value	Additional Information
SAML Assertion Lifetime	1800	
JAAS Context	karaf	Setting this value to karaf lets the STS use the same LDAP that the karaf web console uses for managing users.
Token Issuer	tokenissuer	
Signature Username	tokenissuer	
Encryption Username	server	

Implementation Details

Imported Services

Registered Interface	Availability	Multiple
org.apache.cxf.sts.claims.ClaimsHandler	optional	true
org.apache.cxf.sts.token.validator.TokenValidator	optional	true

Exported Services

None

Security LDAP

Description

The DDF LDAP application allows the user to configure either an embedded or a standalone LDAP server. The provided features contain a default set of schemas and users loaded to help facilitate authentication and authorization testing

Components

Bundle Name	Feature Located In	Description / Link to Bundle Page
ldap-embedded	ldap	Embedded LDAP Configuration

Configuring a Standalone LDAP Server

In some production environments it is suggested that the LDAP server be run separate from the DDF installation. Due to the minimal number of dependencies that the embedded LDAP application requires, this app can be run using a minimal install of DDF that uses much less memory and CPU than a standard installation.

Recommended Steps to Run a Standalone Embedded LDAP instance

1. Obtain and unzip the DDF Kernel (ddf-distribution-kernel-<VERSION>.zip).
2. [Start the distribution](#).
3. When the Kernel has loaded up with the DDF logo at the command prompt, execute

```
la
```

which is short for "list all". Verify that all bundles are *Active*.

Since the kernel does not include all apps, if you were to do a "list" instead of "la," no results would be returned at this point.

4. Finally, deploy the Embedded LDAP App by copying the `ldap-embedded-app-<VERSION>.kar` into the `<DISTRIBUTION_HOME>/deploy` directory. You can verify that the ldap server is installed by checking the DDF log or by performing an `la` and verifying that the OpenDJ bundle is in the *Active* state. Additionally, it should be responding to LDAP requests on the default ports, 1389 and 1636.
5. To perform any of the configurations identified below, the webconsole will need to be installed by executing

```
features:install webconsole
```

Configuration

The configuration options are located on the standard DDF configuration web console under the title *LDAP Server*. It currently contains three configuration options.

Configuration Name	Description
LDAP Port	Sets the port for LDAP (plaintext and StartTLS). 0 will disable the port.
LDAPS Port	Sets the port for LDAPS. 0 will disable the port.
Base LDIF File	Location on the server for a LDIF file. This file will be loaded into the LDAP and overwrite any existing entries. This option should be used when updating the default groups/users with a new ldif file for testing. The LDIF file being loaded may contain any ldap entries (schemas, users, groups..etc). If the location is left blank, the default base LDIF file will be used that comes with DDF.

Trust Certificates

In order for LDAPS to function correctly, it is important that the LDAP Server is configured with a keystore file that trusts the clients it is connecting to and vice versa. Providing your own keystore information for the LDAP Server can be done by doing the following:

1. Navigate to the `/etc/keystores` folder in the kernel distribution folder
2. Find the `serverKeystore.jks` file and replace it with a keystore file valid for your operating environment.
3. If the DDF kernel is running, restart it so the changes will take place

Connecting to a Standalone LDAP Server

DDF instances can connect to an external LDAP server by installing and configuring the `security-sts-server` feature detailed [here](#).

Embedded LDAP Configuration

Description

The Embedded LDAP application contains an LDAP server (OpenDJ version 2.4.6) that has a default set of schemas and users loaded to help facilitate authentication and authorization testing.

Default Settings

Ports

Protocol	Default Port
LDAP	1389
LDAPS	1636

StartTLS	1389
----------	------

Users

LDAP Users

Username	Password	Groups	Description
testuser1	password1		General test user for authentication
testuser2	password2		General test user for authentication
nromanova	password1	avengers	General test user for authentication
lcage	password1	admin, avengers	General test user for authentication, Admin user for karaf
jhowlett	password1	admin, avengers	General test user for authentication, Admin user for karaf
pparker	password1	admin, avengers	General test user for authentication, Admin user for karaf
jdrew	password1	admin, avengers	General test user for authentication, Admin user for karaf
tstark	password1	admin, avengers	General test user for authentication, Admin user for karaf
bbanner	password1	admin, avengers	General test user for authentication, Admin user for karaf
srogers	password1	admin, avengers	General test user for authentication, Admin user for karaf
admin	admin	admin	Admin user for karaf

LDAP Admin

Username	Password	Groups	Attributes	Description
admin	secret			Administrative User for LDAP

Schemas

The default schemas loaded into the LDAP instance are the same defaults that come with OpenDJ.

Schema File Name	Schema Description (http://opendj.forgerock.org/doc/admin-guide/index/chap-schema.html)
00-core.ldif	This file contains a core set of attribute type and objectclass definitions from several standard LDAP documents, including draft-ietf-boreham-numsubordinates, draft-findlay-ldap-groupofentries, draft-furuseth-ldap-untypedobject, draft-good-ldap-changelog, draft-ietf-ldap-subentry, draft-wahl-ldap-adminaddr, RFC 1274, RFC 2079, RFC 2256, RFC 2798, RFC 3045, RFC 3296, RFC 3671, RFC 3672, RFC 4512, RFC 4519, RFC 4523, RFC 4524, RFC 4530, RFC 5020, and X.501.
01-pwpolicy.ldif	This file contains schema definitions from draft-behera-ldap-password-policy, which defines a mechanism for storing password policy information in an LDAP directory server.
02-config.ldif	This file contains the attribute type and objectclass definitions for use with the directory server configuration.
03-changelog.ldif	This file contains schema definitions from draft-good-ldap-changelog, which defines a mechanism for storing information about changes to directory server data.
03-rfc2713.ldif	This file contains schema definitions from RFC 2713, which defines a mechanism for storing serialized Java objects in the directory server.
03-rfc2714.ldif	This file contains schema definitions from RFC 2714, which defines a mechanism for storing CORBA objects in the directory server.
03-rfc2739.ldif	This file contains schema definitions from RFC 2739, which defines a mechanism for storing calendar and vCard objects in the directory server. Note that the definition in RFC 2739 contains a number of errors, and this schema file has been altered from the standard definition in order to fix a number of those problems.
03-rfc2926.ldif	This file contains schema definitions from RFC 2926, which defines a mechanism for mapping between Service Location Protocol (SLP) advertisements and LDAP.
03-rfc3112.ldif	This file contains schema definitions from RFC 3112, which defines the authentication password schema.
03-rfc3712.ldif	This file contains schema definitions from RFC 3712, which defines a mechanism for storing printer information in the directory server.

03-uddiv3.ldif	This file contains schema definitions from RFC 4403, which defines a mechanism for storing UDDIv3 information in the directory server.
04-rfc2307bis.ldif	This file contains schema definitions from the draft-howard-rfc2307bis specification, used to store naming service information in the directory server.
05-rfc4876.ldif	This file contains schema definitions from RFC 4876, which defines a schema for storing Directory User Agent (DUA) profiles and preferences in the directory server.
05-samba.ldif	This file contains schema definitions required when storing Samba user accounts in the directory server.
05-solaris.ldif	This file contains schema definitions required for Solaris and OpenSolaris LDAP naming services.
06-compat.ldif	This file contains the attribute type and objectclass definitions for use with the directory server configuration.

Configuration

Starting / Stopping

The embedded ldap application installs a feature with the name **ldap-embedded**. Installing and Uninstalling this feature will start and stop the embedded ldap server. This will also install a fresh instance of the server each time. If changes need to persist, stopping and starting the embedded-ldap-openssl bundle should be done (rather than installing/uninstalling the feature).

All settings, configurations, and changes made to the embedded LDAP instances are persisted across DDF restarts. If DDF is stopped while the the LDAP feature is installed and started, it will automatically restart with the saved settings on the next DDF start.

Settings

The configuration options are located on the standard DDF configuration web console under the title *LDAP Server*. It currently contains three configuration options.

Configuration Name	Description
LDAP Port	Sets the port for LDAP (plaintext and StartTLS). 0 will disable the port.
LDAPS Port	Sets the port for LDAPS. 0 will disable the port.
Base LDIF File	Location on the server for a LDIF file. This file will be loaded into the LDAP and overwrite any existing entries. This option should be used when updating the default groups/users with a new ldif file for testing. The LDIF file being loaded may contain any ldap entries (schemas, users, groups..etc). If the location is left blank, the default base LDIF file will be used that comes with DDF.

Limitations

Current limitations for the embedded LDAP instances include:

1. Inability to store the LDAP files / storage outside of the DDF installation directory. This results in any LDAP data (i.e. LDAP user information) being lost when the **ldap-embedded** feature is uninstalled.
2. Cannot be run standalone from DDF. In order to run embedded-ldap, the DDF must be started.

External Links

Location to the default base LDIF file in the DDF source code: <https://github.com/codice/ddf/blob/master/ldap/embedded/ldap-embedded-openssl/src/main/resources/default-users.ldif>

OpenDJ Documentation: <http://opendj.forgerock.org/docs.html>

Web Service Security

Introduction

The Web Service Security (WSS) functionality that comes with DDF is integrated throughout the system. This document was made to act as a central point to show how all of the pieces work together and point out where they live inside of 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.

Security Framework

The DDF Security Framework utilizes [Apache Shiro](http://shiro.apache.org/) (<http://shiro.apache.org/>) as the underlying security framework. The classes mentioned in this section will have their full package name listed so that it is 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 (examples: calling a service or returning a metacard). This class was made DDF specific due to the Shiro interface not being able to be added to the Query Request property map.

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 SecurityManager interface makes retrieving Subjects easier and safer.

Implementations of Security Managers:

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

AuthenticationTokens

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.

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 STS. It creates a RequestSecurityToken message from the incoming AuthenticationToken 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/Redaction. 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).

Implementations of Authorizing Realms that come with DDF:

Classname	Description
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.pep.realm.XACMLRealm	This realm delegates the authorization decision to a XACML-based Policy Decision Point (PDP) backend. It creates a XACML 3.0 request and looks on the OSGi framework for any service implementing ddf.security.pdp.api.PolicyDecisionPoint.
ddf.security.pdp.realm.SimpleAuthZRealm	This realm performs the authorization decision without delegating to an external service. It uses the incoming permissions to create a decision.

Auditing

[Auditing](#)

Authentication (AuthN)

Central Authentication Server (CAS)

[CAS SSO Configuration](#)

Authorization (AuthZ)

Service Authorization

[XACML Policy Decision Point \(PDP\)](#)

Resource Authorization

[Redaction and Filtering](#)

Security Token Service

[Security Token Service](#)

Auditing

- CAS (SSO) Authentication
 - Username / Password
 - Sample - Successful login
 - Sample - Failed login
 - PKI Certificate
 - Sample – Successful login
 - Sample – Failed login
- STS Authentication
 - Username / Password

- Sample - Successful login
- Sample - Failed login
- PKI Certificate
 - Sample – Successful login
 - Sample – Failed login
- Binary Security Token (CAS)
 - Sample - Successful Login
 - Sample - Failed Login

Initial set of auditing points.

The Audit Log default location is: DISTRIBUTION_HOME/data/log/security.log

CAS (SSO) Authentication

CAS Authentication Logging was obtained using a CAS war file deployed to a Tomcat application server. Tomcat allows configuration of the log file but by default the logs below were stored in the \$TOMCAT_HOME/logs/catalina.out file.

Username / Password

Sample - Successful login

```
2013-04-24 10:39:45,265 INFO
[org.jasig.cas.authentication.AuthenticationManagerImpl] -
<org.jasig.cas.adaptors.ldap.FastBindLdapAuthenticationHandler successfully
authenticated [username: testuser1]>
2013-04-24 10:39:45,265 INFO
[org.jasig.cas.authentication.AuthenticationManagerImpl] - <Resolved
principal testuser1>
2013-04-24 10:39:45,265 INFO
[org.jasig.cas.authentication.AuthenticationManagerImpl] -
<org.jasig.cas.adaptors.ldap.FastBindLdapAuthenticationHandler@6a4d37e5
authenticated testuser1 with credential [username: testuser1].>
2013-04-24 10:39:45,265 INFO
[com.github.inspektr.audit.support.Slf4jLoggingAuditTrailManager] - <Audit
trail record BEGIN
=====
WHO: [username: testuser1]
WHAT: supplied credentials: [username: testuser1]
ACTION: AUTHENTICATION_SUCCESS
APPLICATION: CAS
WHEN: Wed Apr 24 10:39:45 MST 2013
CLIENT IP ADDRESS: 127.0.0.1
SERVER IP ADDRESS: 127.0.0.1
=====
>
```

Sample - Failed login

```
2013-04-24 10:39:17,443 INFO
[org.jasig.cas.adaptors.ldap.FastBindLdapAuthenticationHandler] - <Failed
to authenticate user testuser1 with error [LDAP: error code 49 - Invalid
Credentials]; nested exception is javax.naming.AuthenticationException:
[LDAP: error code 49 - Invalid Credentials]>
2013-04-24 10:39:17,443 INFO
[org.jasig.cas.authentication.AuthenticationManagerImpl] -
<org.jasig.cas.adaptors.ldap.FastBindLdapAuthenticationHandler failed
authenticating [username: testuser1]>
2013-04-24 10:39:17,443 INFO
[com.github.inspektr.audit.support.Slf4jLoggingAuditTrailManager] - <Audit
trail record BEGIN
=====
WHO: [username: testuser1]
WHAT: supplied credentials: [username: testuser1]
ACTION: AUTHENTICATION_FAILED
APPLICATION: CAS
WHEN: Wed Apr 24 10:39:17 MST 2013
CLIENT IP ADDRESS: 127.0.0.1
SERVER IP ADDRESS: 127.0.0.1
=====
>
```

PKI Certificate

Sample – Successful login

Current testing was done using the OZone certificates as they came with a testAdmin and testUser which were signed by a common CA.


```

2013-04-24 15:13:14,388 INFO
[org.jasig.cas.adaptors.x509.authentication.handler.support.X509Credential
sAuthenticationHandler] - <Successfully authenticated CN=testUser1,
OU=Ozone, O=Ozone, L=Columbia, ST=Maryland, C=US, SerialNumber=4>
2013-04-24 15:13:14,390 INFO
[org.jasig.cas.authentication.AuthenticationManagerImpl] -
<org.jasig.cas.adaptors.x509.authentication.handler.support.X509Credential
sAuthenticationHandler successfully authenticated CN=testUser1, OU=Ozone,
O=Ozone, L=Columbia, ST=Maryland, C=US, SerialNumber=4>
2013-04-24 15:13:14,391 INFO
[org.jasig.cas.authentication.AuthenticationManagerImpl] - <Resolved
principal CN=testUser1, OU=Ozone, O=Ozone, L=Columbia, ST=Maryland, C=US>
2013-04-24 15:13:14,391 INFO
[org.jasig.cas.authentication.AuthenticationManagerImpl] -
<org.jasig.cas.adaptors.x509.authentication.handler.support.X509Credential
sAuthenticationHandler@1e5b04ae authenticated CN=testUser1, OU=Ozone,
O=Ozone, L=Columbia, ST=Maryland, C=US with credential CN=testUser1,
OU=Ozone, O=Ozone, L=Columbia, ST=Maryland, C=US, SerialNumber=4.>
2013-04-24 15:13:14,394 INFO
[com.github.inspektr.audit.support.Slf4jLoggingAuditTrailManager] - <Audit
trail record BEGIN
=====
WHO: CN=testUser1, OU=Ozone, O=Ozone, L=Columbia, ST=Maryland, C=US,
SerialNumber=4
WHAT: supplied credentials: CN=testUser1, OU=Ozone, O=Ozone, L=Columbia,
ST=Maryland, C=US, SerialNumber=4
ACTION: AUTHENTICATION_SUCCESS
APPLICATION: CAS
WHEN: Wed Apr 24 15:13:14 MST 2013
CLIENT IP ADDRESS: 127.0.0.1
SERVER IP ADDRESS: 127.0.0.1
=====
>

```

Sample – Failed login

The failure was simulated using a filter on the x509 credential handler. This filter looks for a certain CN in the certificate chain and will fail if it cannot find a match. The server was setup to trust the certificate via the java truststore, but there were additional requirements put on for logging in. For this test-case the chain it was looking for was "CN=Hogwarts Certifying Authority.+". This was an example from the CAS wiki: <https://wiki.jasig.org/display/CASUM/X.509+Certificates>.

```

2013-04-25 14:15:47,477 DEBUG
[org.jasig.cas.adaptors.x509.authentication.handler.support.X509Credential
sAuthenticationHandler] - <Evaluating CN=testUser1, OU=Ozone, O=Ozone,
L=Columbia, ST=Maryland, C=US, SerialNumber=4>
2013-04-25 14:15:47,478 DEBUG
[org.jasig.cas.adaptors.x509.authentication.handler.support.X509Credential
sAuthenticationHandler] - <.* matches CN=testUser1, OU=Ozone, O=Ozone,
L=Columbia, ST=Maryland, C=US == true>
2013-04-25 14:15:47,478 DEBUG
[org.jasig.cas.adaptors.x509.authentication.handler.support.X509Credential
sAuthenticationHandler] - <CN=Hogwarts Certifying Authority.+ matches
EMAILADDRESS=goss-support@owfgoss.org, CN=localhost, OU=Ozone, O=Ozone,
L=Columbia, ST=Maryland, C=US == false>
2013-04-25 14:15:47,478 DEBUG
[org.jasig.cas.adaptors.x509.authentication.handler.support.X509Credential
sAuthenticationHandler] - <Found valid client certificate>
2013-04-25 14:15:47,478 INFO
[org.jasig.cas.adaptors.x509.authentication.handler.support.X509Credential
sAuthenticationHandler] - <Failed to authenticate
org.jasig.cas.adaptors.x509.authentication.principal.X509CertificateCreden
tials@1795flcc>
2013-04-25 14:15:47,478 INFO
[org.jasig.cas.authentication.AuthenticationManagerImpl] -
<org.jasig.cas.adaptors.x509.authentication.handler.support.X509Credential
sAuthenticationHandler failed to authenticate
org.jasig.cas.adaptors.x509.authentication.principal.X509CertificateCreden
tials@1795flcc>
2013-04-25 14:15:47,478 INFO
[com.github.inspektr.audit.support.Slf4jLoggingAuditTrailManager] - <Audit
trail record BEGIN
=====
WHO:
org.jasig.cas.adaptors.x509.authentication.principal.X509CertificateCreden
tials@1795flcc
WHAT: supplied credentials:
org.jasig.cas.adaptors.x509.authentication.principal.X509CertificateCreden
tials@1795flcc
ACTION: AUTHENTICATION_FAILED
APPLICATION: CAS
WHEN: Thu Apr 25 14:15:47 MST 2013
CLIENT IP ADDRESS: 127.0.0.1
SERVER IP ADDRESS: 127.0.0.1
=====
>

```

STS Authentication

Username / Password

Sample - Successful login

```
14:52:51,168 | INFO | tp1155920414-122 | securityLogger |  
rity.common.audit.SecurityLogger 143 | 253 - security-core-impl -  
2.2.0.RC3-SNAPSHOT | Username [srogers] successfully logged in using LDAP  
authentication. Request IP: 127.0.0.1, Port: 49519
```

Sample - Failed login

```
18:21:06,896 | WARN | qtp460466447-58 | securityLogger |  
rity.common.audit.SecurityLogger 155 | 253 - security-core-impl -  
2.2.0.RC3-SNAPSHOT | Username [srogers] failed LDAP authentication. Request  
IP: 127.0.0.1, Port: 63225
```

PKI Certificate

Sample – Successful login

Sample – Failed login

Binary Security Token (CAS)

Sample - Successful Login

```

15:27:48,098 | INFO | tp1343209378-282 | securityLogger
| rity.common.audit.SecurityLogger 156 | 247 - security-core-api -
2.2.0.RC6-SNAPSHOT | Telling the STS to request a security token on behalf
of the binary security token:
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<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-wssecur
ity-utility-1.0.xsd">U1QtMTctQmw0aGRrS05jaTV3cE82Zm1lVE0tY2FzfGh0dHBzOi8vd
G9rZW5pc3NlZXI6ODk5My9zZXJ2aWNlcY9TZWNlcml0eVRva2VuU2VydmljZQ==</BinarySec
urityToken>
Request IP: 0:0:0:0:0:0:0:1%0, Port: 53363
15:27:48,351 | INFO | tp1343209378-282 | securityLogger
| rity.common.audit.SecurityLogger 156 | 247 - security-core-api -
2.2.0.RC6-SNAPSHOT | Finished requesting security token. Request IP:
0:0:0:0:0:0:0:1%0, Port: 53363

**This message will show when DEBUG is on**
15:27:48,355 | DEBUG | tp1343209378-282 | securityLogger
| rity.common.audit.SecurityLogger 102 | 247 - security-core-api -
2.2.0.RC6-SNAPSHOT | <?xml version="1.0" encoding="UTF-16"?>
<saml2:Assertion>
SAML ASSERTION WILL BE LOCATED HERE

```

Sample - Failed Login

```

10:54:21,772 | INFO | qtp995500086-618 | securityLogger
| rity.common.audit.SecurityLogger 143 | 245 - security-core-commons -
2.2.0.ALPHA5-SNAPSHOT | Telling the STS to request a security token on
behalf of the binary security token:
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<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-wssecur
ity-utility-1.0.xsd">U1QtMjctOU43RUlkNHkzVFoxQmZCb0RIdkItY2Fz</BinarySecur
ityToken>
10:54:22,119 | INFO | qtp995500086-141 | securityLogger
| rity.common.audit.SecurityLogger 143 | 245 - security-core-commons -
2.2.0.ALPHA5-SNAPSHOT | Validating ticket [ST-27-9N7EId4y3TZ1BfBoDHvB-cas]
for service [https://server:8993/services/SecurityTokenService]. Request
IP: 127.0.0.1, Port: 64548
10:54:22,169 | INFO | qtp995500086-141 | securityLogger
| rity.common.audit.SecurityLogger 143 | 245 - security-core-commons -
2.2.0.ALPHA5-SNAPSHOT | Unable to validate CAS token. Request IP:
127.0.0.1, Port: 64548
10:54:22,244 | INFO | qtp995500086-618 | securityLogger
| rity.common.audit.SecurityLogger 143 | 245 - security-core-commons -
2.2.0.ALPHA5-SNAPSHOT | Error requesting the security token from STS at:
https://server:8993/services/SecurityTokenService.

```

CAS SSO Configuration

Introduction

The Web Service Security (WSS) Implementation that comes with DDF was built to be run independent of an SSO or authentication mechanism. Testing out the security functionality of DDF was performed by using the Central Authentication Server (CAS) software. This is a popular SSO appliance and allowed DDF to be tested using realistic use cases. Within this page you will find configurations and settings that we have used to help configure CAS to work within the DDF environment.

General Server Setup and Configuration

The following steps are shown for installing CAS to a Tomcat 7.x server running in Linux. Other configurations, like different versions of Tomcat or other Operating System types, may have slight differences from these steps. Additionally, the setenv.sh file would need to be converted (to a .bat if using Windows).

Installation using DDF CAS WAR

DDF comes with a custom distribution of the CAS web application that comes with LDAP and x509 support configured and built-in. Using this configuration can help save time and setup.

1. Download and Unzip Tomcat Distribution

```
$ unzip apache-tomcat-7.0.39.zip
```

2. Setup Keystores and enable SSL. There are sample configurations located within the security-cas-server-webapp project.
- Copy setenv.sh (ddf-trunk/security/cas/security-cas-server/src/main/resources/tomcat/bin) to TOMCAT/bin

```
$ cp
/ddf-trunk/security/cas/security-cas-server/src/main/resources/t
omcat/setenv.sh apache-tomcat-7.0.39/bin/
```

Make sure to convert setenv.sh to a .bat file if Tomcat is being installed on a windows machine.

- Copy server.xml (ddf-trunk/security/cas/server/security-cas-server-webapp/src/main/resources/tomcat/conf) to TOMCAT/conf

```
$ cp
/ddf-trunk/security/cas/security-cas-server/src/main/resources/t
omcat/conf/server.xml apache-tomcat-7.0.39/conf/
```

- By default, the above files point to TOMCAT/certs/keystore.jks as the default keystore location to use. This file does not come with Tomcat and either needs to be created or the files copied above (setenv.sh and server.xml) need to be modified to point to the correct keystore.

```
$ mkdir apache-tomcat-7.0.39/certs
$ cp keystore.jks apache-tomcat-7.0.39/certs/
```

3. Start Tomcat

```
$ cd apache-tomcat-7.0.39/bin/
$ ./startup.sh
```

Make sure to run startup.bat instead of startup.sh if windows is running on a window machine (also, if you did not convert setenv.sh to a .bat above then startup.bat will not function correctly).

4. Deploy the DDF CAS WAR to Tomcat

```
$ cp /ddf-trunk/security/cas/security-cas-server/target/cas.war
apache-tomcat-7.0.39/webapps/
```

CAS should now be running on the tomcat server. To verify it started without issues, check the tomcat log and look lines similar to the following:

```
Apr 25, 2013 10:55:39 AM org.apache.catalina.startup.HostConfig deployWAR
INFO: Deploying web application archive
/apache-tomcat-7.0.39/webapps/cas.war
2013-04-25 10:55:42,831 INFO
[org.jasig.cas.services.DefaultServicesManagerImpl] - <Loaded 1 services.>
2013-04-25 10:55:43,540 INFO
[org.jasig.cas.util.AutowiringSchedulerFactoryBean] - <Starting Quartz
Scheduler now>
```

CAS will try to authenticate first with x509 (using the keystore provided as the truststore) and failover to LDAP username / password.

The DDF distribution of CAS is configured to use the embedded DDF instance running on localhost. Configuring the LDAP location can be done by modifying the bottom of the cas.properties file located in TOMCAT/webapps/cas/WEB-INF/ after the web application is deployed.

Configuring Existing CAS Installation

For upgrading an existing CAS installation or using the standard CAS web application go to the [Configuring CAS for LDAP](#) page or the [Configuring CAS for X509 User Certificates](#) page for directions on specific configurations that need to be performed.

As part of setting up the server, it is critical to make sure that Tomcat trusts the DDF server certificate and that DDF trusts the certificate from Tomcat. If this is not done correctly, CAS and/or DDF will throw certificate warnings in their logs and will not allow access.

Configuring for DDF

When configuring CAS to integrate with DDF, there are two main configurations that need to be modified. By default, DDF uses 'server' as the hostname for the local DDF instance and 'cas' as the hostname for the CAS server.

CAS Client

The CAS client bundle contains CAS client code that can be used by other bundles when validating and retrieving tickets from CAS. This bundle is extensively used when performing authentication.

When setting up DDF, the 'Server Name' and 'Proxy Callback URL' must be set to the hostname of the local DDF instance.

The 'CAS Server URL' configuration should point to the hostname of the CAS server and should match the SSL certificate that it is using.

CAS Token Validator

The 'CAS Server URL' configuration should point to the hostname of the CAS server and should match the SSL certificate that it is using.

Additional Configuration

Information on each of the CAS-specific bundles that come with DDF as well as their configurations can be found on the [Security CAS](#) application page.

Example Workflow

A sample workflow showing how CAS integrates within the DDF WSS Implementation is described below:

1. User points browser to DDF Query Page.
2. CAS servlet filters are invoked during request
3. (Assuming user is not already signed in) User is redirected to CAS login page.
 - a. For x509 authentication, CAS will try to obtain a certificate from the browser. Most browsers will prompt the user to select a valid certificate to use.
 - b. For username / password authentication, CAS will display a login page.
4. After successful sign-in, user is redirected back to DDF Query page.
5. DDF Query Page obtains the Service Ticket sent from CAS, gets a Proxy Granting Ticket (PGT), and uses that to create a Proxy Ticket for the STS.
6. User fills in search phrase and hits 'search'
7. Security API uses the incoming CAS proxy ticket to create a RequestSecurityToken call to the STS.
8. STS validates the proxy ticket to CAS and creates SAML assertion.

9. Security API returns a Subject class that contains the SAML assertion.
10. Query Page creates a new QueryRequest and adds the Subject into the properties map.

From #10 on, the message is completely de-coupled from CAS and will proceed through the framework properly using the SAML assertion that was created in step 8.

External Links

Official CAS Documentation: <https://wiki.jasig.org/display/CASUM/Home>

Configuring CAS for LDAP

Install / Configure LDAP

DDF comes with an embedded LDAP instance that can be used for testing. During internal testing this LDAP was used extensively.

More information on configuring the LDAP and a list of users and attributes can be found at the [Embedded LDAP Configuration](#) page.

Add *cas-server-support-ldap-3.3.1_1.jar* to CAS

- Copy `thirdparty/cas-server-support-ldap-3.3.1/target/cas-server-support-x509-3.3.1_1.jar` to `$(ozone-widget-framework)/apache-tomcat-$(version)/webapps/cas/WEB-INF/lib/cas-server-support-ldap-3.3.1_1.jar`

Add *spring-ldap-1.2.1_1.jar* to CAS

- Copy `thirdparty/spring-ldap-1.2.1/target/spring-ldap-1.2.1_1.jar` to `$(ozone-widget-framework)/apache-tomcat-$(version)/webapps/cas/WEB-INF/lib/spring-ldap-1.2.1_1.jar`

Modify *developerConfigContext.xml*

- In `$(ozone-widget-framework)/apache-tomcat-$(version)/webapps/cas/WEB-INF/deployerConfigContext.xml`, add the `FastBindLdapAuthenticationHandler` bean definition to the `<list>` in the property stanza with name `authenticationHandlers` of the bean stanza with id `authenticationManager`:

deployerConfigContext.xml

```
<bean id="authenticationManager"
class="org.jasig.cas.authentication.AuthenticationManagerImpl">

    <!-- other property definitions -->

    <property name="authenticationHandlers">
        <list>
            <bean
class="org.jasig.cas.adaptors.ldap.FastBindLdapAuthenticationHandler"
>
                <property name="filter"
value="uid=%u,ou=users,dc=example,dc=com" />
                <property name="contextSource" ref="contextSource" />
            </bean>

            <!-- other bean definitions -->

        </list>
    </property>
</bean>
```


- In `${ozone-widget-framework}/apache-tomcat-${version}/webapps/cas/WEB-INF/deployerConfigContext.xml`, remove the bean stanza with class `ozone3.cas.adapters.UserPropertiesFileAuthenticationHandler` from the `<list>` of the property stanza with name `authenticationHandlers`.
- In `${ozone-widget-framework}/apache-tomcat-${version}/webapps/cas/WEB-INF/deployerConfigContext.xml`, add the `contextSource` bean stanza to the beans stanza:

deployerConfigContext.xml

```
<bean id="contextSource"
class="org.jasig.cas.adapters.ldap.util.AuthenticatedLdapContextSource"
>
  <property name="urls">
    <list>
      <value>ldap://localhost:1389</value>
    </list>
  </property>
  <property name="userDn" value="uid=admin,ou=system"/>
  <property name="password" value="secret"/>
</bean>
```

Ozone Configuration

Ozone was also set up to work in LDAP. This section is here for reference if Ozone is being used in conjunction with CAS. These settings were used for internal testing and should only be used as a reference.

1. Modify `OWFsecurityContext.xml`

- In `${ozone-widget-framework}/apache-tomcat-${version}/lib/OWFsecurityContext.xml`, change the `sec:x509` stanza to the following:

OWFsecurityContext.xml

```
<sec:x509 subject-principal-regex="CN=(.*)",
user-service-ref="ldapUserService" />
```

- In `${ozone-widget-framework}/apache-tomcat-${version}/lib/OWFsecurityContext.xml`, remove the following import:

OWFsecurityContext.xml

```
<import resource="ozone-security-beans/UserServiceBeans.xml"
/>
```

- In `${ozone-widget-framework}/apache-tomcat-${version}/lib/OWFsecurityContext.xml`, add the following import:

OWFsecurityContext.xml

```
<import resource="ozone-security-beans/LdapBeans.xml" />
```

2. Modify `LdapBeans.xml`

- In `${ozone-widget-framework}/apache-tomcat-${version}/lib/ozone-security-beans/LdapBeans.xml`, change the bean stanza

with id contextSource to the following:

LdapBeans.xml

```
<bean id="contextSource"
class="org.springframework.security.ldap.DefaultSpringSecurityContextSource">
    <!-- The URL of the ldap server, along with the base path
that all other ldap path will be relative to -->
    <constructor-arg
value="ldap://localhost:1389/dc=example,dc=com"/>
</bean>
```

- In \${ozone-widget-framework}/apache-tomcat-\${version}/lib/ozone-security-beans/LdapBeans.xml, change the bean stanza with id authoritiesPopulator to the following:

LdapBeans.xml

```
<bean id="authoritiesPopulator"
class="org.springframework.security.ldap.userdetails.DefaultLdapAuthoritiesPopulator">
    <constructor-arg ref="contextSource"/>
    <!-- search base for determining what roles a user has -->
    <constructor-arg value="ou=roles"/>
</bean>
```

- In \${ozone-widget-framework}/apache-tomcat-\${version}/lib/ozone-security-beans/LdapBeans.xml, change the bean stanza with id ldapUserSearch to the following:

LdapBeans.xml

```
<bean id="ldapUserSearch"
class="org.springframework.security.ldap.search.FilterBasedLdapUserSearch">
    <!-- search base for finding User records -->
    <constructor-arg value="ou=users" />
    <constructor-arg value="(uid={0})" /> <!-- filter applied to
entities under the search base in order to find a given user.
this default
searches for an entity with a matching uid -->
    <constructor-arg ref="contextSource" />
</bean>
```

- In \${ozone-widget-framework}/apache-tomcat-\${version}/lib/ozone-security-beans/LdapBeans.xml, change the bean stanza with id userDetailsMapper to the following:

LdapBeans.xml

```
<bean id="userDetailsMapper"
class="ozone.securitysample.authentication.ldap.OWFUserDetailsCo
ntextMapper">
    <constructor-arg ref="contextSource" />
    <!-- search base for finding OWF group membership -->
    <constructor-arg value="ou=groups" />
    <constructor-arg value="(member={0})" /> <!-- filter that
matches only groups that have the given username listed
as a
"member" attribute -->
</bean>
```

3. Modify OWFCASBeans.xml

- In \${ozone-widget-framework}/apache-tomcat-\${version}/lib/ozone-security-beans/OWFCasBeans.xml, change the bean stanza with id casAuthenticationProvider to the following:

OWFCasBeans.xml

```
<bean id="casAuthenticationProvider"
class="org.springframework.security.cas.authentication.CasAuthen
ticationProvider">
    <property name="userService" ref="ldapUserService" />
    <property name="serviceProperties" ref="serviceProperties" />
    <property name="ticketValidator" ref="ticketValidator" />
    <property name="key"
value="an_id_for_this_auth_provider_only" />
</bean>
```

Configuring CAS for X509 User Certificates

Introduction

The follow settings were tested with CAS version 3.3.1. If any issues occur while doing this for newer versions, check the External Links section at the bottom of this page for the CAS documentation on setting up certification authentication.

Add the cas-server-support-x509-3.3.1.jar to CAS

- Copy thirdparty/cas-server-support-x509-3.3.1/target/cas-server-support-x509-3.3.1.jar to apache-tomcat-\${version}/webapps/cas/WEB-INF/lib/cas-server-support-x509-3.3.1.jar

Configuring Web Flow

In apache-tomcat-\${version}/webapps/cas/WEB-INF/login-workflow.xml make the following modifications:

- Remove the XML comments around the **action-state** stanza with id **startAuthenticate**.

startAuthenticate

```
<action-state id="startAuthenticate">
  <action bean="x509Check" />
  <transition on="success" to="sendTicketGrantingTicket" />
  <transition on="error" to="viewLoginForm" />
</action-state>
```

- Modify the **decision-state** stanza with id **renewRequestCheck** as follows.

renewRequestCheck

```
<decision-state id="renewRequestCheck">
  <if test="${externalContext.requestParameterMap['renew'] != ''
&amp;&amp; externalContext.requestParameterMap['renew'] != null}"
then="startAuthenticate" else="generateServiceTicket" />
</decision-state>
```

- Modify the **decision-state** stanza with id **gatewayRequestCheck** as follows.

gatewayRequestCheck

```
<decision-state id="gatewayRequestCheck">
  <if test="${externalContext.requestParameterMap['gateway'] != ''
&amp;&amp; externalContext.requestParameterMap['gateway'] != null
&amp;&amp; flowScope.service != null}" then="redirect"
else="startAuthenticate" />
</decision-state>
```

In `apache-tomcat-${version}/webapps/cas/WEB-INF/cas-servlet.xml` make the following modifications:

- Define the **x509Check** bean.

x509Check

```
<bean
  id="x509Check"
  p:centralAuthenticationService-ref="centralAuthenticationService"

  class="org.jasig.cas.adaptors.x509.web.flow.X509CertificateCredential
sNonInteractiveAction" >
  <property name="centralAuthenticationService"
ref="centralAuthenticationService"/>
</bean>
```

Configuring the Authentication Handler

In `apache-tomcat-${version}/webapps/cas/WEB-INF/deployerConfigContext.xml` make the following modifications:

- In the **list** stanza of the **property** stanza with name **authenticationHandlers** of the **bean** stanza with id **authenticationManager**, add

the **X509CredentialAuthenticationHandler** bean definition.

X509CredentialAuthenticationHandler

```
<bean id="authenticationManager"
  class="org.jasig.cas.authentication.AuthenticationManagerImpl">

  <!-- Other property definitions -->

  <property name="authenticationHandlers">
    <list>

      <!-- Other bean definitions -->

      <bean

class="org.jasig.cas.adaptors.x509.authentication.handler.support.X50
9CredentialsAuthenticationHandler">
        <property name="trustedIssuerDnPattern" value=".*" />
        <!--
          <property name="maxPathLength" value="3" />
          <property name="checkKeyUsage" value="true" />
          <property name="requireKeyUsage" value="true" />
        -->
      </bean>
    </list>
  </property>
</bean>
```

Configuring the Credentials to Principal Resolver

In `apache-tomcat-${version}/webapps/cas/WEB-INF/deployerConfigContext.xml` make the following modifications:

- In the **list** stanza of the **property** stanza with name **credentialsToPrincipalResolver** of the **bean** stanza with id **authenticationManager**, add the **X509CertificateCredentialsToIdentifierPrincipalResolver** bean definition. The pattern in the **value** attribute on the **property** stanza can be modified to suit your needs. This is a simple example that uses the first CN field in the DN as the Principal.

X509CertificateCredentialsToIdentifierPrincipalResolver

```
<bean id="authenticationManager"
  class="org.jasig.cas.authentication.AuthenticationManagerImpl">
  <property name="credentialsToPrincipalResolvers">
    <list>

      <!-- Other bean definitions -->

    </list>

    <bean

      class="org.jasig.cas.adaptors.x509.authentication.principal.X509CertificateCredentialsToIdentifierPrincipalResolver">
        <property name="identifier" value="$OU $CN" />
      </bean>
    </list>
  </property>

  <!-- Other property definitions -->

</bean>
```

In addition to the PrincipalResolver mentioned above, CAS comes with other resolvers that can return different representations of the user identifier. This list was obtained from the official CAS Documentation site linked at the bottom of this page.

Resolver Class	Identifier Output
X509CertificateCredentialsToDistinguishedNamePrincipalResolver	Retrieve the complete distinguished name and use that as the identifier.
X509CertificateCredentialsToIdentifierPrincipalResolver	Transform some subset of the identifier into the ID for the principal.
X509CertificateCredentialsToSerialNumberPrincipalResolver	Use the unique serial number of the certificate.
X509CertificateCredentialsToSerialNumberAndIssuerDNPrincipalResolver	Create a most-likely globally unique reference to this certificate as a DN-like entry, using the CA name and the unique serial number of the certificate for that CA.

Different resolvers should be used depending on the use-case for the server. When performance external attribute lookup (example: Attribute lookup via DIAS) it is necessary to have CAS return the full DN as the identifier and the class X509CertificateCredentialsToDistinguishedNamePrincipalResolver should be used. When using a local LDAP, however, the X509CertificateCredentialsToIdentifierPrincipalResolver class can be used to only return the username that maps directly to the LDAP username.

Default Certificates

- To verify certificate authentication with the default CAS files you must make sure that the included testUser and testAdmin certificates are installed into your web browser. This has only been tested to work with Firefox. These certificates were provided in the Ozone Widget Framework and can be used in development environments.
 - The sample certificate for testUser1 is \${ozone-widget-framework}/apache-tomcat-\${version}/certs/testUser1.p12
 - password: password
 - The sample certificate for testAdmin1 is \${ozone-widget-framework}/apache-tomcat-\${version}/certs/testAdmin1.p12
 - password: password

External Links

For more information on CAS configuration options and what each setting means, go to their documentation page: <https://wiki.jasig.org/display/CASUM/X.509+Certificates>

Certificate Management

DDF uses certificates in a two distinct ways:

1. Transmit and receive encrypted messages.
2. Performing authentication of an incoming user request.

This page details general management operations of using certificates in DDF.

Default Certificates

DDF comes with a number of default keystores that contain certificates. The keystores are used for different services and have different hostnames to denote the services they are being used for.

Alias	Keystore	Truststore	Configuration Location	Usage
server	serverKeystore.jks	serverTruststore.jks	File: <i>etc/org.ops4j.pax.web.cfg</i> File: <i>etc/ws-security/server/encryption.properties</i> File: <i>etc/ws-security/server/signature.properties</i>	Used to secure (SSL) all of the endpoints for DDF. This also includes the admin console and any other web service that is hosted by DDF.
client	clientKeystore.jks	clientTruststore.jks	Web Console: <i>Platform Global Configuration</i> File: <i>etc/ws-security/client/encryption.properties</i> File: <i>etc/ws-security/client/signature.properties</i>	Used for performing outgoing SSL requests. Examples include sending requests to federated sites and verifying tickets with CAS.
tokenissuer	stsKeystore.jks	stsTruststore.jks	File: <i>etc/ws-security/issuer/encryption.properties</i> File: <i>etc/ws-security/issuer/signature.properties</i>	Used to sign STS SAML assertions.

File Management

File management deals with creating and configuring the files that contain the certificates. In DDF these files are generally Java Keystores (jks) and Certificate Revocation Lists (crl). This page gives commands and tools that can be used to perform these operations.

[Cert File Management](#)

Configuration Management

Configuration management deals with configuring DDF to use already made certificates and defining configuration options for the system. This includes configuration certificate revocation and keystores.

[Cert Config Management](#)

Cert Config Management

- [Certificate Revocation Configuration](#)
 - [Enabling Revocation](#)
 - [Adding Revocation to a new Endpoint](#)
 - [Verifying Revocation is taking place](#)

Certificate Revocation Configuration

Enabling Revocation

1. Place the CRL in <ddf.home>/etc/keystores.
2. Uncomment the following line in <ddf.home>/etc/ws-security/server/encryption.properties and replace the filename with the CRL file used in step 1.

```
#org.apache.ws.security.crypto.merlin.x509crl.file=etc/keystores/crlTokenIssuerValid.pem
```

Adding Revocation to a new Endpoint

This guide assumes that the endpoint being created uses CXF and is being started via Blueprint from inside the OSGi container. If other tools are being used the configuration may differ. The [CXF WS-Security Page](#) contains additional information and samples.

- Add the following property to the jaxws endpoint in the endpoint's blueprint.xml:

```
<entry key="ws-security.enableRevocation" value="true"/>
```

- Example xml snippet for the jaxws:endpoint with the property:

```
<jaxws:endpoint id="Test" implementor="#testImpl"
    wsdlLocation="classpath:META-INF/wsdl/TestService.wsdl"
    address="/TestService">

    <jaxws:properties>
        <entry key="ws-security.enableRevocation" value="true"/>
    </jaxws:properties>
</jaxws:endpoint>
```

Verifying Revocation is taking place

A **warning** similar to the following will be displayed in the logs of the source and endpoint showing the exception encountered during certificate validation:

```
11:48:00,016 | WARN | tp2085517656-302 | WSS4JInInterceptor
| ecurity.wss4j.WSS4JInInterceptor 330 | 164 -
org.apache.cxf.cxf-rt-ws-security - 2.7.3 |
org.apache.ws.security.WSSecurityException: General security error (Error
during certificate path validation: Certificate has been revoked, reason:
unspecified)
    at
org.apache.ws.security.components.crypto.Merlin.verifyTrust(Merlin.java:83
8)[161:org.apache.ws.security.wss4j:1.6.9]
    at
org.apache.ws.security.validate.SignatureTrustValidator.verifyTrustInCert(
SignatureTrustValidator.java:213)[161:org.apache.ws.security.wss4j:1.6.9]
    at
org.apache.ws.security.validate.SignatureTrustValidator.validate(Signature
TrustValidator.java:72)[161:org.apache.ws.security.wss4j:1.6.9]
    at
org.apache.ws.security.validate.SamlAssertionValidator.verifySignedAsserti
on(SamlAssertionValidator.java:121)[161:org.apache.ws.security.wss4j:1.6.9]
    at
org.apache.ws.security.validate.SamlAssertionValidator.validate(SamlAssert
ionValidator.java:100)[161:org.apache.ws.security.wss4j:1.6.9]
    at
org.apache.ws.security.processor.SAMLTokenProcessor.handleSAMLToken(SAMLTo
```



```
kenProcessor.java:188)[161:org.apache.ws.security.wss4j:1.6.9]
    at
org.apache.ws.security.processor.SAMLTokenProcessor.handleToken(SAMLTokenP
rocessor.java:78)[161:org.apache.ws.security.wss4j:1.6.9]
    at
org.apache.ws.security.WSSecurityEngine.processSecurityHeader(WSSecurityEn
gine.java:396)[161:org.apache.ws.security.wss4j:1.6.9]
    at
org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor.handleMessage(WSS4JInI
nterceptor.java:274)[164:org.apache.cxf.cxf-rt-ws-security:2.7.3]
    at
org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor.handleMessage(WSS4JInI
nterceptor.java:93)[164:org.apache.cxf.cxf-rt-ws-security:2.7.3]
    at
org.apache.cxf.phase.PhaseInterceptorChain.doIntercept(PhaseInterceptorCha
in.java:271)[123:org.apache.cxf.cxf-api:2.7.3]
    at
org.apache.cxf.transport.ChainInitiationObserver.onMessage(ChainInitiation
Observer.java:121)[123:org.apache.cxf.cxf-api:2.7.3]
    at
org.apache.cxf.transport.http.AbstractHTTPDestination.invoke(AbstractHTTPD
estination.java:239)[130:org.apache.cxf.cxf-rt-transports-http:2.7.3]
    at
org.apache.cxf.transport.servlet.ServletController.invokeDestination(Servl
etController.java:218)[130:org.apache.cxf.cxf-rt-transports-http:2.7.3]
    at
org.apache.cxf.transport.servlet.ServletController.invoke(ServletControlle
r.java:198)[130:org.apache.cxf.cxf-rt-transports-http:2.7.3]
    at
org.apache.cxf.transport.servlet.ServletController.invoke(ServletControlle
r.java:137)[130:org.apache.cxf.cxf-rt-transports-http:2.7.3]
    at
org.apache.cxf.transport.servlet.CXFNonSpringServlet.invoke(CXFNonSpringSe
rvlet.java:158)[130:org.apache.cxf.cxf-rt-transports-http:2.7.3]
    at
org.apache.cxf.transport.servlet.AbstractHTTPServlet.handleRequest(Abstrac
tHTTPServlet.java:243)[130:org.apache.cxf.cxf-rt-transports-http:2.7.3]
    at
org.apache.cxf.transport.servlet.AbstractHTTPServlet.doPost(AbstractHTTPSe
rvlet.java:163)[130:org.apache.cxf.cxf-rt-transports-http:2.7.3]
    at
javax.servlet.http.HttpServlet.service(HttpServlet.java:713)[52:org.apache
.geronimo.specs.geronimo-servlet_2.5_spec:1.1.2]
    at
org.apache.cxf.transport.servlet.AbstractHTTPServlet.service(AbstractHTTPSe
rvlet.java:219)[130:org.apache.cxf.cxf-rt-transports-http:2.7.3]
    at
org.eclipse.jetty.servlet.ServletHolder.handle(ServletHolder.java:547)[63:
org.eclipse.jetty.servlet:7.5.4.v20111024]
    at
org.eclipse.jetty.servlet.ServletHandler.doHandle(ServletHandler.java:480)
[63:org.eclipse.jetty.servlet:7.5.4.v20111024]
    at
```

```
org.ops4j.pax.web.service.jetty.internal.HttpServiceServletHandler.doHandle(
HttpServiceServletHandler.java:70)[73:org.ops4j.pax.web.pax-web-jetty:1.
0.11]
    at
org.eclipse.jetty.server.handler.ScopedHandler.handle(ScopedHandler.java:1
19)[61:org.eclipse.jetty.server:7.5.4.v20111024]
    at
org.eclipse.jetty.security.SecurityHandler.handle(SecurityHandler.java:520
)[62:org.eclipse.jetty.security:7.5.4.v20111024]
    at
org.eclipse.jetty.server.session.SessionHandler.doHandle(SessionHandler.ja
va:227)[61:org.eclipse.jetty.server:7.5.4.v20111024]
    at
org.eclipse.jetty.server.handler.ContextHandler.doHandle(ContextHandler.ja
va:941)[61:org.eclipse.jetty.server:7.5.4.v20111024]
    at
org.ops4j.pax.web.service.jetty.internal.HttpServiceContext.doHandle(HttpS
erviceContext.java:117)[73:org.ops4j.pax.web.pax-web-jetty:1.0.11]
    at
org.eclipse.jetty.servlet.ServletHandler.doScope(ServletHandler.java:409)[
63:org.eclipse.jetty.servlet:7.5.4.v20111024]
    at
org.eclipse.jetty.server.session.SessionHandler.doScope(SessionHandler.jav
a:186)[61:org.eclipse.jetty.server:7.5.4.v20111024]
    at
org.eclipse.jetty.server.handler.ContextHandler.doScope(ContextHandler.jav
a:875)[61:org.eclipse.jetty.server:7.5.4.v20111024]
    at
org.eclipse.jetty.server.handler.ScopedHandler.handle(ScopedHandler.java:1
17)[61:org.eclipse.jetty.server:7.5.4.v20111024]
    at
org.eclipse.jetty.server.handler.HandlerCollection.handle(HandlerCollectio
n.java:149)[61:org.eclipse.jetty.server:7.5.4.v20111024]
    at
org.eclipse.jetty.server.handler.HandlerWrapper.handle(HandlerWrapper.java
:110)[61:org.eclipse.jetty.server:7.5.4.v20111024]
    at
org.eclipse.jetty.server.Server.handle(Server.java:349)[61:org.eclipse.jet
ty.server:7.5.4.v20111024]
    at
org.eclipse.jetty.server.HttpConnection.handleRequest(HttpConnection.java:
441)[61:org.eclipse.jetty.server:7.5.4.v20111024]
    at
org.eclipse.jetty.server.HttpConnection$RequestHandler.content(HttpConnect
ion.java:936)[61:org.eclipse.jetty.server:7.5.4.v20111024]
    at
org.eclipse.jetty.http.HttpParser.parseNext(HttpParser.java:893)[57:org.ec
lipse.jetty.http:7.5.4.v20111024]
    at
org.eclipse.jetty.http.HttpParser.parseAvailable(HttpParser.java:218)[57:org.eclipse.jetty.http:7.5.4.v20111024]
    at
org.eclipse.jetty.server.BlockingHttpConnection.handle(BlockingHttpConnect
```

```
ion.java:50)[61:org.eclipse.jetty.server:7.5.4.v20111024]
    at
org.eclipse.jetty.server.bio.SocketConnector$ConnectorEndPoint.run(SocketC
onconnector.java:245)[61:org.eclipse.jetty.server:7.5.4.v20111024]
    at
org.eclipse.jetty.server.ssl.SslSocketConnector$SslConnectorEndPoint.run(S
slSocketConnector.java:663)[61:org.eclipse.jetty.server:7.5.4.v20111024]
    at
org.eclipse.jetty.util.thread.QueuedThreadPool.runJob(QueuedThreadPool.jav
a:598)[55:org.eclipse.jetty.util:7.5.4.v20111024]
    at
org.eclipse.jetty.util.thread.QueuedThreadPool$3.run(QueuedThreadPool.java
:533)[55:org.eclipse.jetty.util:7.5.4.v20111024]
    at java.lang.Thread.run(Thread.java:662)[:1.6.0_33]
Caused by: java.security.cert.CertPathValidatorException: Certificate has
been revoked, reason: unspecified
    at
sun.security.provider.certpath.PKIXMasterCertPathValidator.validate(PKIXMa
sterCertPathValidator.java:139)[:1.6.0_33]
    at
sun.security.provider.certpath.PKIXCertPathValidator.doValidate(PKIXCertPa
thValidator.java:330)[:1.6.0_33]
    at
sun.security.provider.certpath.PKIXCertPathValidator.engineValidate(PKIXCe
rtPathValidator.java:178)[:1.6.0_33]
    at
java.security.cert.CertPathValidator.validate(CertPathValidator.java:250)[
:1.6.0_33]
    at
```

```
org.apache.ws.security.components.crypto.Merlin.verifyTrust(Merlin.java:81
4)[161:org.apache.ws.security.wss4j:1.6.9]
... 45 more
```

Cert File Management

- Tools Used
- General Certificates
 - Creating a CA Key and Certificate
 - Create a key pair
 - Use the key to sign the CA certificate
 - Using the CA to Sign Certificates
 - Generate a private key and a Certificate Signing Request (CSR)
 - Sign the certificate by the CA
- Java Keystore (JKS)
 - Creating a new Keystore/Truststore with an existing Certificate and Private Key
 - Import into a Java Keystore (JKS)
 - Put the private key and the certificate into one file
 - Put the private key and the certificate in a PKCS12 keystore
 - Import the PKCS12 keystore into a Java keystore (JKS)
 - Change the alias
- Certificate Revocation List (CRL)
 - Creating a Certificate Revocation List (CRL)
 - Revoking a Certificate and Creating a New CRL that Contains the Revoked Certificate
 - Viewing a CRL

Tools Used

- openssl
 - Windows users can use: [openssl for windows](#)
- The standard java **keytool** certificate management utility.
- **Portecle** can be used for **keytool** operations if a GUI is preferred over a command line interface.

General Certificates

Creating a CA Key and Certificate

The following shows how to create a root CA to sign certificates.

Create a key pair

```
$> openssl genrsa -aes128 -out root-ca.key 1024
```

Use the key to sign the CA certificate

```
$> openssl req -new -x509 -days 3650 -key root-ca.key -out root-ca.crt
```

Using the CA to Sign Certificates

The following shows how to sign a certificate for the tokenissuer user by a CA.

Generate a private key and a Certificate Signing Request (CSR)

```
$> openssl req -newkey rsa:1024 -keyout tokenissuer.key -out
tokenissuer.req
```

Sign the certificate by the CA

```
$> openssl ca -out tokenissuer.crt -infile tokenissuer.req
```

Java Keystore (JKS)

Creating a new Keystore/Truststore with an existing Certificate and Private Key

Using the private key, certificate, and CA certificate one can create a new keystore containing the data from the new files.

```
cat client.crt >> client.key
openssl pkcs12 -export -in client.key -out client.p12
keytool -importkeystore -srckeystore client.p12 -destkeystore
clientKeystore.jks -srcstoretype pkcs12 -alias 1
keytool -changealias -alias 1 -destalias client -keystore
clientKeystore.jks
keytool -importcert -file ca.crt -keystore clientKeystore.jks -alias "ca"
keytool -importcert -file ca-root.crt -keystore clientKeystore.jks -alias
"ca-root"
```

The truststore can be created with only the use of the CA certificate. Based on the concept of CA signing, the CA should be the only entry needed in the truststore.

```
keytool -import -trustcacerts -alias "ca" -file ca.crt -keystore
truststore.jks
keytool -import -trustcacerts -alias "ca-root" -file ca-root.crt -keystore
truststore.jks
```

Using the certificate one can create a PEM file from it as it is the format that some applications use.

```
openssl x509 -in client.crt -out client.der -outform DER
openssl x509 -in client.der -inform DER -out client.pem -outform PEM
```

Import into a Java Keystore (JKS)

The following shows how to import a PKCS12 keystore generated by openssl into a Java keystore (JKS).

Put the private key and the certificate into one file

```
$> cat tokenissuer.crt >> tokenissuer.key
```

Put the private key and the certificate in a PKCS12 keystore

```
$> openssl pkcs12 -export -in tokenissuer.key -out tokenissuer.p12
```

Import the PKCS12 keystore into a Java keystore (JKS)

```
$> keytool -importkeystore -srckeystore tokenissuer.pl2 -destkeystore  
stsKeystore.jks -srcstoretype pkcs12 -alias 1
```

Change the alias

```
$> keytool -changealias -alias 1 -destalias tokenissuer
```

Certificate Revocation List (CRL)

Creating a Certificate Revocation List (CRL)

Using the CA create in the above steps, one can create a CRL in which the tokenissuer's certificate is valid.

```
$> openssl ca -gencrl -out crl-tokenissuer-valid.pem
```

Revoking a Certificate and Creating a New CRL that Contains the Revoked Certificate

```
$> openssl ca -revoke tokenissuer.crt  
  
$> openssl ca -gencrl -out crl-tokenissuer-revoked.pem
```

Viewing a CRL

The following command will list the serial numbers of the revoked certificates.

```
$> openssl crl -inform PEM -text -noout -in crl-tokenissuer-revoked.pem
```

Encryption Service

The Encryption Service and encryption command, 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 one to encrypt plain text. This is useful if one wishes to display password fields in a GUI.

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

```
ddf@local>security:encrypt myPasswordToEncrypt  
  
bR9mJpDVo8bTRwqGwIFxHJ5yFJzatKwjXjIo/8USWm8=
```

Redaction and Filtering

Where it happens:

Redaction and filtering is performed in a Post Query Plugin that occurs after a query has been performed.

How it happens:

Each Metacard result will contain security attributes pulled from the metadata record after being processed by a PostQueryPlugin that populates this attribute. The security attribute is a HashMap containing a set of keys that map to lists of values. The Metacard will then be processed by a filter/redaction plugin that creates a KeyValueCollectionPermission from the Metacard's security attribute. This permission is then checked against the user Subject in order to determine if the Subject has the correct claims in order to view that Metacard. The decision of whether or not to filter/redact* the Metacard eventually ends up in the hands of the installed PDP. (features:install security-pdp-java OR features:install security-pdp-xacml). Whichever PDP is being used will return back a decision and the Metacard will either be filtered/redacted or allowed to pass through.

*The default setting is to redact records.

The security attributes populated on the Metacard are completely dependent on the type of the Metacard. Each type of Metacard must have its own PostQueryPlugin that reads the metadata being returned and populates the Metacard's security attribute. If either the subject or resource (Metacard) permissions are missing during redaction, that resource will be redacted.

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>
```

So with 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 KeyValueCollectionPermission 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.

Redaction Policies:

Setting up a policy is different depending on which PDP implementation is installed. The security-pdp-java implementation is the simplest PDP to use, so it will be covered here:

1. Open the Web Console*
2. Click on the Configuration tab
3. Click on the Authz Security Settings configuration
4. Add any roles that are allowed to access protected services
5. Add any SOAP actions that are not to be protected by the PDP
6. Add any attribute mappings necessary to map between Subject claims and Metacard values
 - a. For example, the above example would require 2 Match All mappings of: claim1=entry1 and claim2=entry2
 - b. Match One mappings would contain: claim3=entry3 and claim4=entry4

*See the page at [Security PDP AuthZ Realm](#) for a description of the configuration page.

With the security-pdp-java feature configured in this way, the above Metacard would be displayed to the user.

The XACML PDP is explained in more detail in [XACML Policy Decision Point \(PDP\)](#). It is up to the administrator to write a XACML policy capable of returning the correct response message. The Java based PDP should perform adequately in most situations. It is also possible to install both the security-pdp-java and security-pdp-xacml features at the same time. The system could be configured in this way in order to allow the Java PDP to handle most cases and only have XACML policies to handle more complex situations than what the Java PDP is designed for. Just keep in mind that this would be a very complex configuration with both PDP's installed and it should only be done if you understand what you're doing.

How to redact a new type of Metacard:

All that is necessary to enable redaction/filtering on a new type of record is implementing a PostQueryPlugin that is able to read the String metadata contained within the Metacard record. The plugin must set the security attribute to a map of list of values extracted from the metacard. Note that in DDF, there is no default plugin that populates the security attribute on the Metacard. A plugin must be created to populate these fields in order for redaction/filtering to work correctly.

Example Redacted Record:


```

<?xml version="1.0" encoding="UTF-8"?>
<metacard xmlns="urn:catalog:metacard"
xmlns:ns2="http://www.opengis.net/gml"
xmlns:ns3="http://www.w3.org/1999/xlink"
xmlns:ns4="http://www.w3.org/2001/SMIL20/"
xmlns:ns5="http://www.w3.org/2001/SMIL20/Language"
ns2:id="99f494b22f4341e9a3ba3ef6a5fe8734">
  <type>ddf.metacard</type>
  <source>ddf.distribution</source>
  <stringxml name="metadata">
    <value>
      <meta:Resource xmlns:meta="...">
        <meta:identifier meta:qualifier="http://metadata/noaccess"
meta:value="REDACTED"/>
        <meta:title>REDACTED</meta:title>
        <meta:creator>
          <meta:Organization>
            <meta:name>REDACTED</meta:name>
          </meta:Organization>
        </meta:creator>
        <meta:subjectCoverage>
          <meta:Subject>
            <meta:category meta:label="REDACTED"/>
          </meta:Subject>
        </meta:subjectCoverage>
        <meta:security SEC:controls="A B" SEC:group="X" SEC:origin="USA"/>
      </meta:Resource>
    </value>
  </stringxml>
  <string name="resource-uri">
    <value>catalog://metadata/noaccess</value>
  </string>
  <string name="title">
    <value>REDACTED</value>
  </string>
  <string name="resource-size">
    <value>REDACTED</value>
  </string>
</metacard>

```

Security Token Service

- [Description](#)
- [Using the STS](#)
 - [Standalone Installation](#)
- [STS Claims Handlers](#)
 - [Adding a Custom Claims Handler](#)
- [STS WS-Trust WSDL Document](#)
- [Example Request & Responses for a SAML Assertion](#)
 - [RequestSecurityToken](#)
- [Known Issues](#)

Description

The STS provides a DDF that a system can request a SAML v2.0 assertion from.

The STS is an extension of Apache CXF-STS. It is a SOAP web service that utilizes WS-Security policies. 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.

Using the STS

Once installed the STS can be used to request SAML v2.0 assertions via a SOAP web service request. Out of the box it supports authentication from existing SAML tokens, CAS Proxy tickets, Username/Password, and x509 certificates. It also supports retrieving claims using LDAP.

Standalone Installation

The STS cannot currently be installed on a kernel distribution of DDF. To run a STS-only DDF installation, uninstall the catalog components that are not being used. The list below shows features which can be uninstalled to minimize the runtime size of DDF in an STS-only mode. This list is not a comprehensive list of every feature that can be uninstalled but rather a list of the larger components that can be uninstalled without impacting the STS functionality.

Unneeded Features
catalog-core-standardframework
catalog-solr-embedded-provider
catalog-opensearch-endpoint
catalog-opensearch-souce
catalog-rest-endpoint

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 in additional claims handlers that can handle other types of external services that store user attributes.

Adding a Custom Claims Handler

Description:

A claim is an additional piece of data about a principal 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:

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

Steps:

The following outlines the steps required to add a custom claims handler to the STS:

- The new claims handler must implement the `org.apache.cxf.sts.claims.ClaimsHandler` 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);

}

```

- 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>

```

- Deploy the bundle.

If the new claims handler is hitting an external service that is secured with SSL you may have to add the root CA of the external site to the DDF trustStore and add a valid certificate into the DDF keyStore so that it can encrypt messages that will be accepted by the external service. For more information on certificates, check out the [Configuring a Java Keystore for Secure Communications](#) page.

STS WS-Trust WSDL Document

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-wssecur
ity-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/CancelF
inal" 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/IssueF
inal" 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/RenewFi
nal" 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/Validat
eFinal" 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/KETFina
l" 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/KeyExchan
geToken"/>
        <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/RequestCo
llection"/>
        <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>
```

Example Request & Responses for a SAML Assertion

The DDF STS offers many different ways to requesting 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.

RequestSecurityToken

[UsernameToken Sample Request/Response](#)

[X.509 Token Sample Request/Response](#)

[BinarySecurityToken \(CAS\) Sample Request/Response](#)

Known Issues

BinarySecurityToken (CAS) Sample Request/Response

BinarySecurityToken (CAS) Sample Request / Response

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-a
      4a7-8a5ce243a7cb</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-wssecu
      rity-secext-1.0.xsd"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecu
      rity-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</w
st: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:Addres
s>
  </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-wssecur
ity-utility-1.0.xsd">U1QtMTQtYUtmcDYxcFRtS0FxZGlpVDMzOWMtY2FzfGh0dHBzOi8vd
G9rZW5pc3NlZXI6ODk5My9zZXJ2aWNlcy9TZWNlcm10eVRva2VuU2VydmJjZQ==</BinarySec
urityToken>
  </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</w
st:KeyType>
  <wst:UseKey>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:X509Data>
        <ds:X509Certificate>
```

MIIC5DCCAk2gAwIBAgIJAKj7ROPHjo1yMA0GCSqGSIB3DQEBCwUAMIGKMQswCQYDVQQGEwJVUzEQ
MA4GA1UECAwHQXJpem9uYTERMA8GA1UEBwwIR29vZHllYXlXGDAWBgNVBAoMD0xvY2toZWVkieE1h
cnRpbjENMASGA1UECwwESTRDRTEPMA0GA1UEAwwGY2xpZW50MRwwGgYJKoZIhvcNAQkBFglpNGNl
QGxtY28uY29tMB4XDTEyMDYyMDE5NDMwOVoxDTIyMDYxODE5NDMwOVowgYoxCzAJBgNVBAYTA1VT
MRAwDgYDVQQIDAdBcm16b25hMREwDwYDVQQHDAhHb29keWVhcjEYMBYGA1UECgwPTG9ja2hlZWQg
TWFydGluMQ0wCwYDVQQLDARJNENFMQ8wDQYDVQQDDAZjbGllbnQxHDAaBgkqhkiG9w0BCQEWDWk0
Y2VAbG1jby5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAIPhxCBLYE7xfDLcITS9SsPG
4Q04Z6S32/+TriGsRgpGTj/7GuMG7oJ98m6Ws5cTYl7nyunyHTkZuP7rBzy4esDIHheyx18EgdSJ
vvACgGVCnEmHndkf9bWU1AOfNaxW+vZwljUkRUVdkhPbPdPwOcMdKg/SsLSNjZfsQIjoWd4rAgMB
AAGjUDBOMB0GA1UdDgQWBQBQx11VLtYXLvFGpFdHnhlNW9+lxBDafBgNVHSMEGDAWgBQx11VLtYXL
vFGpFdHnhlNW9+lxBDAMBgNVHRMEBTADAQH/MA0GCSqGSIB3DQEBCwUAA4GBAHys2OI0K6yVXzyS
sKcv2fmfw6XCICGTnyA7BodAjYoqq6wD+33dHJUCFDqye7AWdcivuc7RWJt9jnlfJZKIm2BHcDTR
Hhk6CvjJl4Gf40WQdeMHox8U8b0diq7Iy5Ravx+zRg7SdiyJUqFYjRh/O5tywXRTl+freI3bwAN0
L6tQ
</ds:X509Certificate>
 </ds:X509Data>
 </ds:KeyInfo>
</wst:UseKey>
<wst:Renewing/>

```
</wst:RequestSecurityToken>
</soap:Body>
</soap:Envelope>
```

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-b
      08b-0689ffa9c93e</MessageID>
    <To
      xmlns="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/add
      ressing/anonymous</To>
    <RelatesTo
      xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:60652909-faca-4e4a-a
      4a7-8a5ce243a7cb</RelatesTo>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecu
      rity-secext-1.0.xsd"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecu
      rity-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-wssecu
      rity-utility-1.0.xsd"
      xmlns:ns3="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecu
      rity-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:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>

  <ds:SignatureValue>h+NvkgXGdQtca3/eKebhAKgG38tHp3i2n5uLLy8xXXIg02qyKgEP0FC
owp2LiYlsQU9YjKfSwCUbH3WR6jhbAv9zj29CE+ePfEny7MeXvgNl3wId+vcHqti/DGghhgtO2
Mbx/tyXlBhHQUwKrlcHajxHeecwmvV7D85NMdV48tI=</ds:SignatureValue>

  <ds:KeyInfo>
    <ds:X509Data>

<ds:X509Certificate>MIIDmjCCAwoGAWIBAgIBBDANBgkqhkiG9w0BAQQFADB1MQswCQYDVQ
QGEwJUVUzEQMA4GA1UECBMH
QXJpem9uYTERMA8GA1UEBxMIR29vZHllYXlxEADAQBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0
V4
YW1wbGUxEDAOBgNVBAStB0V4YW1wbGUxQzAJBgNVBAMTAkNBMB4XDTEzMjQwOTE4MzcwMVoxDTIz
MDQwNzE4MzcwMVowgaYxCzAJBgNVBAYTAlVTMRAwDgYDVQQQIEwdbcm16b25hMREwDwYDVQQHEW
hH
b29keWVhcjEQMA4GA1UEChMHRXhhbXBsZTEQMA4GA1UEChMHRXhhbXBsZTEQMA4GA1UECzMHRX
hh
bXBsZTEUMBIGAlUEAxMLdG9rZW5pc3NlZXlxEjAKBgkqhkiG9w0BCQEFwF3Rva2VuaXNzdWVyQG
V4
YW1wbGUuYy29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDDfktP8Lrp9rTfRibKdgtxt
N9
uB44diiIqq3JOzDGfDhGLu6mjpuH0lhrKItv42hB0hhmH7lS9ipiaQCIPVfgIG63MB7fa5dBrf
GF
G69vFrU1Lfi7IvsVVsNrtAEQljoMmw9sxs3SUSRQX+bD8jq7Uj1hpoF7DdqpV8Kb0COOGwIDAQ
AB
o4IBBjCCAQIwCQYDVR0TBAlwADAsBg1ghkgBhvCAQ0EHxYdT3BlblNTTCBHZW5lcmF0ZWQgQ2
Vy
dG1maWNhdGUwHQYDVR0OBByEFd1mHviop2Tc4HaNu8yPXR6GqWP1MIGnBgNVHSMEgZ8wgZyAFB
cn
en6/j05DzaVwORwrteKc7TZOOxmkdzB1MQswCQYDVQQGEwJUVUzEQMA4GA1UECBMHQXJpem9uYT
ER
MA8GA1UEBxMIR29vZHllYXlxEADAQBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0V4YW1wbGUxED
```

```

AO
BgNVBAsTB0V4YW1wbGUxCzAJBgNVBAMTAkNBggkAwXk70cwO7gwwDQYJKoZIhvcNAQEEBQADgY
EA
PiTX5kYXwdhmi jutSkrObKpRbQkvkzcyZlO6VrAxRQ+eFeN6NyuyhgYy5K6l/sIWdaGou5iJO
Qx
2pQYWxl v8Klyl0W22IfEAXYv/epiO89hpdACryuDjpioXI/X8TAWvRwLKL2lDk3k2b+eyCgA00
++
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>MIIC5DCCAk2gAwIBAgIJAKj7ROPHjolyMA0GCSqGSIB3DQEBCwUAMI
GKMQswCQYDVQQGEWJVUzEQ
MA4GA1UECAwHQXJpem9uYTERMA8GA1UEBwwIR29vZHl1YXlXGDAWBgNVBAoMD0xvY2toZWVkie
1h
cnRpbjENMAsGA1UECwwESTRDRTEPMA0GA1UEAwwGY2xpZW50MRwwGgYJKoZIhvcNAQkBFglpNG
Nl
QGxtY28uY29tMB4XDTEyMDYyMDE5NDMwOVVoXDTIyMDYxODE5NDMwOVowgYoxCzAJBgNVBAYTAl
VT
MRAwDgYDVQQIDAdBcm16b25hMREwDwYDVQQHDAhHb29keWVhcjEYMBYGA1UECgwPTG9ja2hlZW
Qg
TWfydGluMQ0wCwYDVQQLDARJNENFMQ8wDQYDVQQDDAZjbGllbnQxHDAABgkqhkiG9w0BCQEW
DWk0
Y2VAbG1jb25jb20wgZ8wDQYJKoZIhvcNAQEEBQADgY0AMIGJAoGBAIPhxCBLyE7xfDLcITS9Ss
PG
4Q04Z6S32/+TriGsRgpGTj/7GuMG7oJ98m6Ws5cTYl7nyunyHTkZuP7rBzy4esDIHheyx18Egd
SJ
vvACgGVCnEmHndkf9bWU1AOfNaxW+vZwljUkRUVdkhPbPdPwOcMdKg/SsLSNjZfsQIjoWd4rAg
MB
AAGjUDBOMB0GA1UdDgQWBbQx11VLtYXLvFGpFdHnhlNW9+lxBDafBgNVHSMEGDAWgBQx11VLtY
XL
vFGpFdHnhlNW9+lxBDAMBgNVHRMEBTADAQH/MA0GCSqGSIB3DQEBCwUAA4GBAHYs2OI0K6yVXz
ys
sKcv2fmfw6XCICGTnyA7B0dAjYoqq6wD+33dHJUcFDqye7AWdcivuc7RWJt9jnlfJZKIm2BHcD
TR
Hhk6CvjJ14Gf40WQdeMHoX8U8b0diq7Iy5Ravx+zRg7SdiyJUqFYjRh/O5tywXRTl+freI3bwa
N0
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/SecurityTokenService</saml2:A
udience>
    </saml2:AudienceRestriction>
  </saml2:Conditions>
  <saml2:AuthnStatement AuthnInstant="2013-04-29T18:35:11.392Z">
    <saml2:AuthnContext>

<saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:unspeci
fied</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:wssell="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1
.1.xsd"
wssell:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-prof
ile-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:wssell="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1
.1.xsd"
wssell:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-prof
ile-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</wsa:Addres
s>
        </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 Sample Request/Response

UsernameToken Sample Request / Response

These request and responses messages were taken from the same thread and should be viewed as a corresponding pair.

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-wssecu
      rity-secext-1.0.xsd"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecu
      rity-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:Security>

    <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue</ws
    a:Action>
    <wsa:MessageID>uuid:albba87b-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.oas
          is-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.oas
          is-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"
          />
        </t:Claims>
      </wst:SecondaryParameters>
    </wst:RequestSecurityToken>
  </soap:Body>
</soap:Envelope>
```

```
        <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:OnBehalfOf>
            <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-tok
en-profile-1.0#PasswordText">password1</wsse:Password>
            </wsse:UsernameToken>
        </wst:OnBehalfOf>

    <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue</w
st: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

```
<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-a
      53c-13d960e3b6e8</MessageID>
    <To
      xmlns="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/add
      ressing/anonymous</To>
    <RelatesTo
      xmlns="http://www.w3.org/2005/08/addressing">uuid:albba87b-0f00-46cc-975f-
      001391658cbe</RelatesTo>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecu
      rity-secext-1.0.xsd"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecu
      rity-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-wssecu
      rity-utility-1.0.xsd"
      xmlns:ns3="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecu
      rity-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:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>

  <ds:SignatureValue>X5Kzd54PrKiIlgVV2XxzCmWFRzHROYbF7hU6zxbEhSLMR0AWS9R7Me3e
pq9lXqeOwvIDDbwmE/oJNC7vI0fIw/rqXkx4aZsY5a5nbAs7f+aXF9TGdk82x2eNhNGYpViq0Y
ZJfsJ5WSyMtG8w5nRekmDMY9oTLsHG+Y/OhJDEWq58=</ds:SignatureValue>

  <ds:KeyInfo>
    <ds:X509Data>

      <ds:X509Certificate>MIIDmJCCAwoGAWIBAgIBBDANBgkqhkiG9w0BAQQFAADB1MQswCQYDVQ
QGEwJVUzEQMA4GA1UECBMH
QXJpem9uYTERMA8GA1UEBxMIR29vZHllYXlxEADAQBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0
V4
YW1wbGUxEDAOBgNVBAwTB0V4YW1wbGUxCzAJBgNVBAMTAkNBMB4XDTEzMDQwOTE4MzcwMVoXDTE
Iz
MDQwNzE4MzcwMVowgaYxCzAJBgNVBAYTA1VTMRAwDgYDVQQIEWdBcm16b25hMREwDwYDVQQHEW
hH
b29keWVhcjEQMA4GA1UEChMHRXhhbXBsZTEQMA4GA1UEChMHRXhhbXBsZTEQMA4GA1UECXMHRX
hh
bXBsZTEUMBGA1UEAxMLdG9rZW5pc3N1ZXlXJjAkBgkqhkiG9w0BCQEFwF3Rva2VuaXNzdWVyQG
V4
YW1wbGUuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDDfktpA8Lrp9rTfRibKdgtxt
N9
uB44diiIqq3JOzDGfDhGLu6mjpuHO1hrKIItv42hBOhhmH7lS9ipiaQCIPVfgIG63MB7fa5dBrf
GF
G69vFrU1Lfi7IvsVVsNrtAEQljOMmw9sxS3SUSRQX+bD8jq7Uj1hpoF7DdqpV8Kb0COOGwIDAQ
AB
o4IBBjCCAQIwCQYDVR0TBAlwADAsBg1ghkgBhvCAQ0EHxYdT3BlblNTTCBHZW5lcmF0ZWQgQ2
Vy
dGlmaWNhdGUwHQYDVR0OBBYEFd1mHviop2Tc4HaNu8yPXR6GqWP1MIGnBgNVHSMEgZ8wgZyAFB
cn
en6/j05DzaVwORwrteKc7TZ0oXmkdzB1MQswCQYDVQQGEwJVUzEQMA4GA1UECBMHQXJpem9uYT
ER
MA8GA1UEBxMIR29vZHllYXlxEADAQBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0V4YW1wbGUxED
```

```
AO
BgNVBAsTB0V4YWlwbGUxCzAJBgNVBAMTAkNBggkAwXk7OcwO7gwwDQYJKoZIhvcNAQEEBQADgY
EA
PiTX5kYXwdhmi jutSkrObKpRbQkvkzcyZl06VrAxRQ+eFeN6NyuyhgYy5K6l/sIWdaGou5iJO
Qx
2pQYWxl v8Klyl0W22IfEAXYv/epi089hpdACryuDJpioXI/X8TAwvRwLKL2lDk3k2b+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:ac:classes:unspeci
fied</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:wssell="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1
.1.xsd"
wssell:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-prof
ile-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:wssell="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1
.1.xsd"
wssell:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-prof
ile-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:Address>
    </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 Token Sample Request/Response

X.509 Token Sample Request / Response

This request and response pair was done using the <http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey> type and contains the key to use inside of the SOAP message.

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:c0c43e1e-0264-4018-9
      a58-d1fda4332ab3</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-wssecu
      rity-secext-1.0.xsd"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecur
      ity-utility-1.0.xsd" soap:mustUnderstand="1">
      <wsu:Timestamp wsu:Id="TS-3">
        <wsu:Created>2013-04-29T18:08:05.556Z</wsu:Created>
        <wsu:Expires>2013-04-29T18:18:05.556Z</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:SecondaryParameters>
          <t:TokenType
            xmlns:t="http://docs.oasis-open.org/ws-sx/ws-trust/200512">http://docs.oas
            is-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.oas
            is-open.org/ws-sx/ws-trust/200512/PublicKey</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/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</w
st: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>MIIC5DCCAk2gAwIBAgIJAKj7ROPHjo1yMA0GCSqGSIb3DQEBCwUAMI
GKMQswCQYDVQQGEwJVUzEQ
MA4GA1UECAwHQXJpem9uYTERMA8GA1UEBwwIR29vZHllYXlXGDAWBgNVBAoMD0xvY2toZWVkieE
1h
cnRpbjENMAsGA1UECwwESTRDRTEPMA0GA1UEAwwGY2xpZW50MRwwGgYJKoZIhvcNAQkBFglpNG
Nl
QGxtY28uY29tMB4XDTEyMDYyMDE5NDMwOV0xODUyMDYxODE5NDMwOVowgYoxCzAJBgNVBAYTAI
VT
MRAwDgYDVQQQIDAdBcm16b25hMREwDwYDVQQHDAhHb29keWVhcjEYMBYGA1UECgwPTG9ja2hlZW
Qg
TWFydGluMQ0wCwYDVQQQLDARJNENFMQ8wDQYDVQQDDAZjbGllbnQxHDAaBgkqhkiG9w0BCQEW
DW
k0
Y2VAbG1jb25jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAIPhxCBLYE7xfDLcITS9Ss
PG
4Q04Z6S32/+TriGsRgpGTj/7GuMG7oJ98m6Ws5cTYl7nyunyHTkZuP7rBzy4esDIHheyx18Egd
SJ
vvACgGVCnEmHndkf9bWU1AOfNaxW+vZwljUkRUVdkhPbPdPwOcMdkG/SsLSNjzfsQIjoWd4rAg
MB
AAGjUDBOMB0GA1UdDgQWBQx11VLtYXLvFGpFdHnhlNW9+lxBDafBgNVHSMEGDAWgBQx11VLtY
XL
vFGpFdHnhlNW9+lxBDAMBgNVHRMEBTADAQH/MA0GCSqGSIb3DQEBCwUAA4GBAHys2OI0K6yVXz
yS
sKcv2fmfw6XCICGTnyA7BodAjYoqq6wD+33dHJUCFDqye7AWdcivuc7RWJt9jnlfJZKIm2BHcD
TR
Hhk6CvjJ14Gf40WQdeMHoX8U8b0diq7Iy5Ravx+zRg7SdiyJUqFYjRh/05tywXRT1+freI3bwA
N0
L6tQ</ds:X509Certificate>
```

```
</ds:X509Data>  
</ds:KeyInfo>  
</wst:UseKey>  
<wst:Renewing/>
```

```
</wst:RequestSecurityToken>
</soap:Body>
</soap:Envelope>
```

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:c535ff31-6241-48c1-a
      07c-d8ee3e4385e7</MessageID>
    <To
      xmlns="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/add
      ressing/anonymous</To>
    <RelatesTo
      xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:c0c43e1e-0264-4018-9
      a58-d1fda4332ab3</RelatesTo>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecu
      rity-secext-1.0.xsd"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecur
      ity-utility-1.0.xsd" soap:mustUnderstand="1">
      <wsu:Timestamp wsu:Id="TS-2">
        <wsu:Created>2013-04-29T18:08:39.472Z</wsu:Created>
        <wsu:Expires>2013-04-29T18:13:39.472Z</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-wssecur
      ity-utility-1.0.xsd"
      xmlns:ns3="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecur
      ity-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:xsi="http://www.w3.org/2001/XMLSchema-instance"
            ID="_33A83A3092E9D441A613672589194562"
            IssueInstant="2013-04-29T18:08:39.456Z" 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="#_33A83A3092E9D441A613672589194562">
    <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#" />
    </ds:Transforms>
    <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
    <ds:DigestValue>/pQNGarh9sgdKWv/O+6DGOAj4eg=</ds:DigestValue>
  </ds:Reference>
</ds:SignedInfo>

<ds:SignatureValue>CoF328yMctb2LyMyh2bZhUZi0/bJBZEO+jpDVlK/VGQHBTlU3paqcdE
v1qcJSPsllNiX5ZW7Qbrl4hyIkLf0Aj5F6zYGVgC/JUtXxmLxeDPDO2IY5LV1RUDy3f8c8Bxf9
++xCRu7h3iarDRKDHCK5eGSfC+dIjcX36OSAS0URCw=</ds:SignatureValue>

<ds:KeyInfo>
  <ds:X509Data>

<ds:X509Certificate>MIIDmjCCAwoGAWIBAgIBBDANBgkqhkiG9w0BAQQFADB1MQswCQYDVQ
QGEwJVUzEQMA4GA1UECBMH
QXJpem9uYTERMA8GA1UEBxMIR29vZHllYXlxeDAOBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0
V4
YW1wbGUxEDAOBgNVBAwTB0V4YW1wbGUxCzAJBgNVBAMTAkNBMB4XDTEzMDQwOTE4MzcwMVoxDT
Iz
MDQwNzE4MzcwMVowgaYxCzAJBgNVBAYTA1VTMRAwDgYDVQQIEWdBcm16b25hMREwDwYDVQQHEW
hH
b29keWVhcjEQMA4GA1UEChMHXhbbXBsZTEQMA4GA1UEChMHXhbbXBsZTEQMA4GA1UECXMHRX
hh
bXBsZTEUMBGA1UEAxMLdG9rZW5pc3N1ZXlxejAkBgkqhkiG9w0BCQEFWF3Rva2VuaXNzdWVyQG
V4
YW1wbGUuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDDfktP8Lrp9rTfRibKdgtxt
N9
uB44diiIqq3JOzDGfDhGLu6mjpuH01hrKItv42hBOhhmH7lS9ipiaQCIPvfgIG63MB7fa5dBrf
GF
G69vFrU1Lfi7IvsVVsNrtAEQljOMmw9sxS3SUSRQX+bd8jq7Uj1hpoF7DdqpV8Kb0COOGwIDAQ
AB
o4IBBjCCAQIwCQYDVR0TBAlwADAsBg1ghkgBhvCAQ0EHxYdT3BlblNTTCBHZW5lcmF0ZWQgQ2
Vy
dG1maWNhdGUwHQYDVR0OBByEFd1mHviop2Tc4HaNu8yPXR6GqWP1MIGnBgNVHSMEgZ8wgZyAFB
cn
en6/j05DzaVwORwrteKc7TZ0oXmkdzB1MQswCQYDVQQGEwJVUzEQMA4GA1UECBMHQXJpem9uYT
ER
MA8GA1UEBxMIR29vZHllYXlxeDAOBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0V4YW1wbGUxED
AO
BgNVBAwTB0V4YW1wbGUxCzAJBgNVBAMTAkNBggkAwXk70cw07gwwDQYJKoZIhvcNAQEEBQADgY
EA
PiTX5kYXwdhmi jutSkrObKpRbQkvkzcyZl06VrAxRQ+eFeN6NyuyhgYy5K6l/sIWdaGou5iJO
```

```
Qx
2pQYWxlV8Klyl0W22IfEAXYv/epi089hpdACryuDjpioXI/X8TAwvRwLKL2lDk3k2b+eyCgA00
++
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">1.2.840.113549.1.9.1=#160d693463
65406c6d636f2e636f6d,CN=server,OU=I4CE,O=Lockheed
Martin,L=Goodyear,ST=Arizona,C=US</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>MIIC5DCCAk2gAwIBAgIJAKj7ROPHjo1yMA0GCSqGSIb3DQEBCwUAMI
GKMQswCQYDVQQGEwJVUzEQ
MA4GA1UECAwHQXJpem9uYTERMA8GA1UEBwwIR29vZHllYXlXGDAWBgNVBAoMD0xvY2toZWVkie
1h
cnRpbjENMAsGA1UECwwESTRDRTEPMA0GA1UEAwwGY2xpZW50MRwwGgYJKoZIhvcNAQkBFglpNG
Nl
QGxtY28uY29tMB4XDTEyMDYyMDE5NDMwOVVoXDTIyMDYxODE5NDMwOVowGyYoxCzAJBgNVBAYTA1
VT
MRAwDgYDVQQIDAdBcm16b25hMREwDwYDVQQHDAhHb29keWVhcjEYMBYGA1UECgwPTG9ja2hlZW
Qg
TWFyZGluMQ0wCwYDVQQLDARJNENFMQ8wDQYDVQQDDAZjbGl1bnQxHDAaBgkqhkiG9w0BCQEWDA
k0
Y2VAbG1jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAIPhxCBLYE7xfDLcITS9Ss
PG
4Q04Z6S32/+TriGsRgpGTj/7GuMG7oJ98m6Ws5cTYl7nyunyHTkZuP7rBzy4esDIHheyx18Egd
SJ
vvACgGVCnEmHndkf9bWU1AOfNaxW+vZwljUkRUVdkhPbPdPwOcMdKg/SsLSNjZfsQIjoWd4rAg
MB
AAGjUDBOMB0GA1UdDgQWBQBQx11VLtYXlVFGpFdHnhlNW9+lxBDafBgNVHSMEGDAWgBQx11VLtY
XL
vFGpFdHnhlNW9+lxBDAMBgNVHRMEBTADAQH/MA0GCSqGSIb3DQEBCwUAA4GBAHys2OI0K6yVXz
yS
sKcv2fmfw6XCICGTnyA7BodAjYoqq6wD+33dHJUCFDqye7AWdcivuc7RWJt9jnlfJZKIm2BHcD
TR
Hhk6CvjJ14Gf40WQdeMHoX8U8b0diq7Iy5Ravx+zRg7SdiyJUqFYjRh/05tywXRT1+freI3bwA
N0
L6tQ</ds:X509Certificate>
    </ds:X509Data>
    </ds:KeyInfo>
    </saml2:SubjectConfirmationData>
    </saml2:SubjectConfirmation>
    </saml2:Subject>
    <saml2:Conditions NotBefore="2013-04-29T18:08:39.457Z">
```

```

NotOnOrAfter="2013-04-29T18:38:39.457Z">
    <saml2:AudienceRestriction>

<saml2:Audience>https://server:8993/services/QueryService</saml2:Audience>
    </saml2:AudienceRestriction>
</saml2:Conditions>
<saml2:AuthnStatement AuthnInstant="2013-04-29T18:08:39.457Z">
    <saml2:AuthnContext>

<saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:unspeci
fied</saml2:AuthnContextClassRef>
    </saml2:AuthnContext>
    </saml2:AuthnStatement>
</saml2:Assertion>
</RequestedSecurityToken>
<RequestedAttachedReference>
    <ns3:SecurityTokenReference
xmlns:wssell="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1
.1.xsd"
wssell:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-prof
ile-1.1#SAMLV2.0">
    <ns3:KeyIdentifier
ValueType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1
#SAMLID">_33A83A3092E9D441A613672589194562</ns3:KeyIdentifier>
    </ns3:SecurityTokenReference>
</RequestedAttachedReference>
<RequestedUnattachedReference>
    <ns3:SecurityTokenReference
xmlns:wssell="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1
.1.xsd"
wssell:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-prof
ile-1.1#SAMLV2.0">
    <ns3:KeyIdentifier
ValueType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1
#SAMLID">_33A83A3092E9D441A613672589194562</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:Address>
    </wsa:EndpointReference>
</wsp:AppliesTo>
<Lifetime>
    <ns2:Created>2013-04-29T18:08:39.467Z</ns2:Created>
    <ns2:Expires>2013-04-29T18:38:39.467Z</ns2:Expires>
</Lifetime>
</RequestSecurityTokenResponse>
</RequestSecurityTokenResponseCollection>
</soap:Body>

```

```
</soap:Envelope>
```

XACML Policy Decision Point (PDP)

- [Configuration Notes](#)
- [Creating a Policy](#)
 - [Actions](#)
 - [Attributes](#)
 - [Subject](#)
 - [Resource](#)
- [Example Requests and Responses](#)
 - [Policy](#)
 - [Service Authorization](#)
 - [Allowed Query](#)
 - [Denied Query](#)
 - [Metacard Authorization](#)
 - [Subject Permitted](#)
 - [Subject Denied](#)

Configuration Notes

After unzipping the DDF distribution, place the desired XACML pollicy in the **<distribution root>/etc/pdp/policies** directory. This is the directory in which the PDP will look for XACML policies every 60 seconds. A sample XACML policy is located at the end of this page.

Information on specific bundle configurations and names can be found on the [Security PDP](#) application page.

Creating a Policy

This document assumes familiarity with the XACML schema and will not go into detail on the XACML language. There are some DDF-specific items that need to be considered when creating a policy so that it works with the XACMLRealm. When creating a policy, a target is used to specify that a certain action should be run only for a type of request. Targets can be used on both the main Policy element and also any individual rules. Generally targets are geared towards the actions that are set in the request.

Actions

For DDF these actions are populated by various components in the security API. The actions and where they are populated is described in the table below:

Operation	Action-id Value	Component Setting the action	Description
Filtering / Redaction	filer	security-pdp-xacmlrealm	When performing any redaction or filtering the XACMLRealm will set the action-id to be "filer".
Service	<SOAPAction>	security-pep-interceptor	If the PEP Interceptor is added to any SOAP-based web services for service authorization the action-id will be the SOAPAction of the incoming request. This allows the XACML policy to have specific rules for individual services within the system.

These are only the action-id values that are currently created by the components that come with DDF. Additional components can be made and added to DDF that can specify specific action-ids.

In the Examples section below, the policy has specified targets for the above type of calls. For the Filtering / Redaction code, the target was set for "filer" while for the Service validation code the targets were geared toward two services: query, and LocalSiteName. In production environment these actions for service authorization will generally be full URNs that are describe within the SOAP WSDL.

Attributes

Attributes for the XACML request are populated from information in the calling subject as well as the resource being checked against.

Subject

The attributes for the subject are obtained from the SAML claims and populated within the XACMLRealm as individual attributes under the **urn:oa sis:names:tc:xacml:1.0:subject-category:access-subject** category. The name of the claim is used for the **Attributeld** value. Examples of these items being populated are available the end of this section.

Resource

The attributes for resources are obtained through the permissions process. When checking permissions, the XACMLRealm retrieves a list of permissions that should be checked against the subject. These permissions are populated outside of realm and should be populated with the security attributes located in the metacard security property. When the permissions are of a key-value type, the key being used is populated as the **AttributeId** value under the **urn:oasis:names:tc:xacml:3.0:attribute-category:resource** category.

Example Requests and Responses

The following items are a sample request, response, and the corresponding policy. For the XACML PDP the request is made by the XACML Realm (security-pdp-xacmlrealm), passed to the XACML Processing Engine (security-pdp-xacmlprocessor) that reads the policy and outputs a response.

Policy

This is the sample policy that was used for the following sample request and responses. The policy was made to handle the following actions: filter, query, and LocalSiteName. The filter action is used to compare subject's SUBJECT_ACCESS attributes to metacard's RESOURCE_ACCESS attributes. The query and LocalSiteName actions differ as they are used to perform service authorization. For a query, the user must be associated with the country of code ATA (Antarctica) and a LocalSiteName action can be performed by anyone.

Policy

```
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
PolicyId="xpath-target-single-req"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:
permit-overrides" Version="1.0">
  <PolicyDefaults>
    <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
  </PolicyDefaults>
  <Target>
    <AnyOf>
      <Allof>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">filter</AttributeValue>
          <AttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        </Match>
      </Allof>
    </AnyOf>
      <Allof>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">query</AttributeValue>
          <AttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        </Match>
      </Allof>
    </AnyOf>
      <Allof>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">LocalSiteName</Attribut
```

```

eValue>
  <AttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
  </Match>
</Allof>
</AnyOf>
</Target>
<Rule Effect="Permit" RuleId="permit-filter">
  <Target>
    <AnyOf>
      <Allof>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">filter</AttributeValue>
          <AttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
          </Match>
        </Allof>
      </AnyOf>
    </Target>
    <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-subset">
        <AttributeDesignator AttributeId="RESOURCE_ACCESS"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        <AttributeDesignator AttributeId="SUBJECT_ACCESS"
Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        </Apply>
      </Condition>
    </Rule>
    <Rule Effect="Permit" RuleId="permit-action">
      <Target>
        <AnyOf>
          <Allof>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">ATA</AttributeValue>
              <AttributeDesignator
AttributeId="http://www.opm.gov/feddata/CountryOfCitizenship"
Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
              </Match>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">query</AttributeValue>
              <AttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"

```

```
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
  </Match>
</AllOf>
<AllOf>
  <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">LocalSiteName</Attribut
eValue>
      <AttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
```

```
</Rule>
<Rule Effect="Deny" RuleId="deny-read"/>
</Policy>
```

Service Authorization

Allowed Query

Request

```
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
ReturnPolicyIdList="false" CombinedDecision="false">
  <Attributes
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">query</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes
Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
IncludeInResult="false">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Test
User</AttributeValue>
    </Attribute>
    <Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"
IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">users</AttributeValue>
    </Attribute>
    <Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"
IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">admin</AttributeValue>
    </Attribute>
    <Attribute AttributeId="SUBJECT_ACCESS" IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">A</AttributeValue>
    </Attribute>
    <Attribute AttributeId="SUBJECT_ACCESS" IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">B</AttributeValue>
    </Attribute>
    <Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameide
```

```
ntifier" IncludeInResult="false">
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">testuser1</AttributeVal
ue>
  </Attribute>
  <Attribute
    AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname" IncludeInResult="false">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Test
User</AttributeValue>
  </Attribute>
  <Attribute AttributeId="http://www.opm.gov/feddata/CountryOfCitizenship"
IncludeInResult="false">
    <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#string">ATA</AttributeValue>
```

```
</Attribute>
</Attributes>
</Request>
```

Response

```
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
  <Result>
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```

Denied Query

Request

```
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
ReturnPolicyIdList="false" CombinedDecision="false">
  <Attributes
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">query</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes
Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
IncludeInResult="false">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Test
User USA</AttributeValue>
    </Attribute>
    <Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"
IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">users</AttributeValue>
    </Attribute>
    <Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"
IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">admin</AttributeValue>
    </Attribute>
    <Attribute AttributeId="SUBJECT_ACCESS" IncludeInResult="false">
      <AttributeValue
```

```
DataType="http://www.w3.org/2001/XMLSchema#string">A</AttributeValue>
  </Attribute>
  <Attribute AttributeId="SUBJECT_ACCESS" IncludeInResult="false">
    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">B</AttributeValue>
  </Attribute>
  <Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier" IncludeInResult="false">
  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">testuser1</AttributeValue>
  </Attribute>
  <Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname" IncludeInResult="false">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Test
User</AttributeValue>
  </Attribute>
  <Attribute AttributeId="http://www.opm.gov/feddata/CountryOfCitizenship"
IncludeInResult="false">
  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">USA</AttributeValue>
```

```
</Attribute>
</Attributes>
</Request>
```

Response

```
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
  <Result>
    <Decision>Deny</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```

Metacard Authorization

Subject Permitted

All of the resource's RESOURCE_ACCESS attributes were matched with the Subject's SUBJECT_ACCESS attributes.

Request

```
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
ReturnPolicyIdList="false" CombinedDecision="false">
  <Attributes
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">filter</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes
Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
IncludeInResult="false">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Test
User</AttributeValue>
    </Attribute>
    <Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"
IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">users</AttributeValue>
    </Attribute>
    <Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"
IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">admin</AttributeValue>
```



```
</Attribute>
<Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier" IncludeInResult="false">
  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">testuser1</AttributeValue>
</Attribute>
<Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname" IncludeInResult="false">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Test
User</AttributeValue>
</Attribute>
<Attribute AttributeId="SUBJECT_ACCESS" IncludeInResult="false">
  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">A</AttributeValue>
</Attribute>
<Attribute AttributeId="SUBJECT_ACCESS" IncludeInResult="false">
  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">B</AttributeValue>
</Attribute>
<Attribute AttributeId="http://www.opm.gov/feddata/CountryOfCitizenship"
IncludeInResult="false">
  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">ATA</AttributeValue>
</Attribute>
</Attributes>
<Attributes
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
  <Attribute AttributeId="RESOURCE_ACCESS" IncludeInResult="false">
    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">A</AttributeValue>
```

```
</Attribute>
</Attributes>
</Request>
```

Response

```
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
  <Result>
    <Decision>Deny</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```

Subject Denied

The resource had an additional RESOURCE_ACCESS attribute 'C' that the subject did not have.

Request

```
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
ReturnPolicyIdList="false" CombinedDecision="false">
  <Attributes
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">filter</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes
Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
IncludeInResult="false">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Test
User</AttributeValue>
    </Attribute>
    <Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"
IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">users</AttributeValue>
    </Attribute>
    <Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"
IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">admin</AttributeValue>
    </Attribute>
  </Attributes>
```

```
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname" IncludeInResult="false">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Test
User</AttributeValue>
</Attribute>
<Attribute
AttributeId="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier" IncludeInResult="false">
  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">testuser1</AttributeValue>
</Attribute>
<Attribute AttributeId="SUBJECT_ACCESS" IncludeInResult="false">
  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">A</AttributeValue>
</Attribute>
<Attribute AttributeId="SUBJECT_ACCESS" IncludeInResult="false">
  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">B</AttributeValue>
</Attribute>
<Attribute AttributeId="http://www.opm.gov/feddata/CountryOfCitizenship"
IncludeInResult="false">
  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">ATA</AttributeValue>
</Attribute>
</Attributes>
<Attributes
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
  <Attribute AttributeId="RESOURCE_ACCESS" IncludeInResult="false">
    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">A</AttributeValue>
</Attribute>
    <Attribute AttributeId="RESOURCE_ACCESS" IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">B</AttributeValue>
</Attribute>
    <Attribute AttributeId="RESOURCE_ACCESS" IncludeInResult="false">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">C</AttributeValue>
```

```
</Attribute>
</Attributes>
</Request>
```

Response

```
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
  <Result>
    <Decision>Deny</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```

Expansion Service

Overview

The Expansion Service and its corresponding expansion-related commands, provides an easy way for developers to add expansion capabilities to DDF during user attributes 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)	Rules (original->new)	
key1	value1	replacement1
	value2	replacement2
	value3	replacement3
key2	value1	replacement1
	value2	replacement2

The examples below use the following collection of rule sets:

Key (Attribute)	Rules (original -> new)	
Location	Goodyear	Goodyear AZ
	AZ	AZ USA
	CA	CA USA
Title	VP-Sales	VP-Sales VP Sales
	VP-Engineering	VP-Engineering VP Engineering

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.

Instances and Configuration

It is expected that multiple instances of the expansion service will be running at the same time. Each instance of the service defines a unique

property useful for retrieving specific instances of the expansion service. The following table lists the two pre-defined instances used by DDF for expanding user attributes and metacard attributes respectively.

Property Name	Value	Description
mapping	security.user.attribute.mapping	This instance is configured with rules that expand user's attribute values for security checking.
mapping	security.metacard.attribute.mapping	This instance is configured with rules that expand the metacard's security attributes before comparing with the user's attributes.

Each instance of the expansion service can be configured using a configuration file. The configuration file can have three different types of lines:

- comments - any line prefixed with the '#' character is ignored as a comment (for readability, blank lines are also ignored)
- attribute separator - a line starting with 'separator=' defines the attribute separator string.
- rule - all other lines are assumed to be rules defined in a string format <key>:<original value>:<new value>

The following configuration file defines the rules shown above in the example table (using the space as a separator):

```
# This defines the separator that will be used when the expansion string
contains multiple
# values - each will be separated by this string. The expanded string will
be split at the
# separator string and each resulting attributed added to the attribute set
(duplicates are
# suppressed). No value indicates the default value of ' ' (space).
separator=

# The following rules define the attribute expansion to be performed. The
rules are of the
# form:
#     <attribute name>:<original value>:<expanded value>
# The rules are ordered, so replacements from the first rules may be found
in the original
# values of subsequent rules.
Location:Goodyear:Goodyear AZ
Location:AZ:AZ USA
Location:CA:CA USA
Title:VP-Sales:VP-Sales VP Sales
Title:VP-Engineering:VP-Engineering VP Engineering
```

Expansion Commands

Title	Namespace	Description
DDF::Security::Expansion::Commands	security	The expansion commands provide detailed information about the expansion rules in place and the ability to see the results of expanding specific values against the active rule set.

Expansion Commands

```
security:expand          security:expansions
```

Command Descriptions

Command	Description
expand	Runs the expansion service on the provided data returning the expanded value
expansions	Dumps the ruleset for each active expansion service.

Expansion Command Examples and Explanation

security:expansions

The **security:expansions** command dumps the ruleset for each active expansion service. It takes no arguments and displays each rule on a separate line in the form: <attribute name> : <original string> : <expanded string>. The following example shows the results of executing the **expansions** command with no active expansion service.

```
ddf@local>security:expansions
No expansion services currently available.
```

After installing the expansions service and configuring it with an appropriate set of rules, the **expansions** command will provide output similar to the following:

```
ddf@local>security:expansions
Location : Goodyear : Goodyear AZ
Location : AZ : AZ USA
Location : CA : CA USA
Title : VP-Sales : VP-Sales VP Sales
Title : VP-Engineering : VP-Engineering VP Engineering
```

security:expand

The **security:expand** command runs the expansion service on the provided data. It takes an attribute and an original value, expands the original value using the current expansion service and rule set and dumps the results. For the rule set shown above, the **expand** command produces the following results:

```
ddf@local>security:expand Location Goodyear
[Goodyear, USA, AZ]

ddf@local>security:expand Title VP-Engineering
[VP-Engineering, Engineering, VP]

ddf@local>expand Title "VP-Engineering Manager"
[VP-Engineering, Engineering, VP, Manager]
```

Configuring DDF with WSS using standalone authentication servers.

DDF uses CAS as its single sign-on service, and LDAP and STS to keep track of users and user attributes. CAS, LDAP, and STS are integral, interconnected components of the DDF security scheme, and all can be installed on a local DDF instance with only a few feature installs (with the exception of the [CAS installation](#), which requires Apache Tomcat to run). Setting up these authentication components to run externally, however, is a little more nuanced, so this page will provide step-by-step instructions detailing the configuration process.

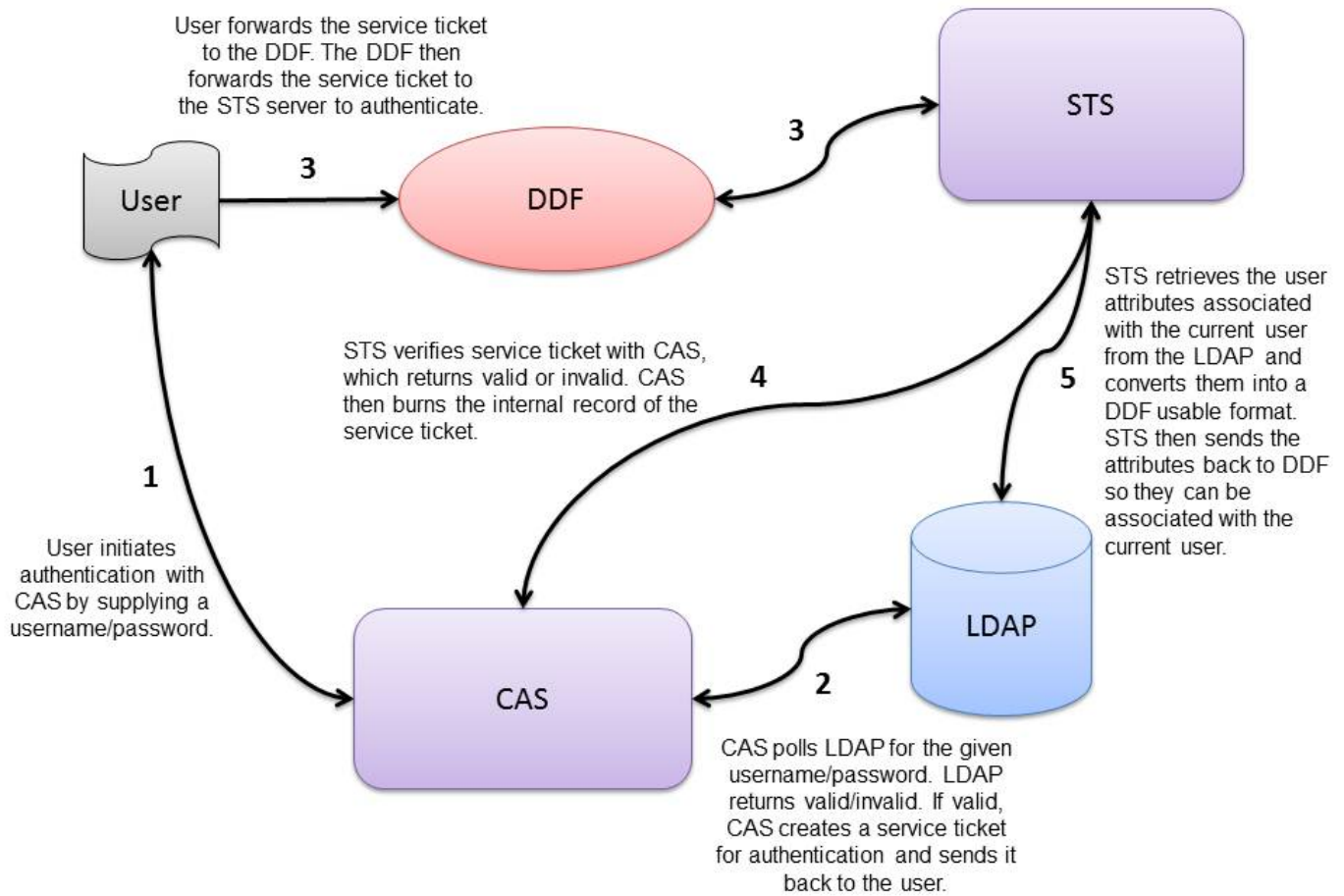
This document assumes that you have a keystore for each of the services / servers. If you are using different keystore names, substitute the name in this document with the desired name for your setup. For this document, the following is used:

Server	Keystore File	Comments
CAS	keystore.jks	Used on the CAS Tomcat server.

STS	stsKeystore.jks	Used to sign SAML and also as incoming connections
DDF	serverKeystore.jks	Server used for incoming connections
	clientKeystore.jks	Client used for outgoing connections

Figure 1: Login Authentication Scheme

Distributed WSS.pptx



Configuring CAS, LDAP, and STS

It is implied that the three authentication components below will be installed onto three separate servers, and as such it is important to keep track of the DNS hostnames used in each for certificate authentication purposes.

LDAP

LDAP is used to maintain a list of trusted DDF users and the attributes associated with them. It interacts with both CAS and the STS: the former uses LDAP to create session information, and the latter queries LDAP for user attributes and converts them to SAML Claims.

1. Obtain and unzip the DDF Kernel (ddf-distribution-kernel-<VERSION>.zip).
2. Start the distribution.
3. Deploy the Embedded LDAP App by copying the ldap-embedded-app-<VERSION>.kar into the <DISTRIBUTION_HOME>/deploy directory. You can verify that the ldap server is installed by checking the DDF log or by performing an `la` command and verifying that the OpenDJ bundle is in the Active state. Additionally, it should be responding to LDAP requests on the default ports, 1389 and 1636.
4. Copy your environment's Java Keystore File into the `$(DISTRIBUTION)/etc/keystores` folder, making sure it overwrites the folder's existing `serverKeystore.jks` file.

It is very important that the keystore file used in the process is set up to trust the hostnames used by CAS and STS, or else the user will run into certificate authentication issues.

CAS

CAS is used for SSO authentication purposes. Unlike LDAP and STS, CAS can not be run as a DDF bundle and instead must be run through Apache Tomcat.

1. Follow the instructions on the [CAS installation](#) page in order to get Tomcat/CAS installed and configured (like with LDAP above, the keystore.jks file used must trust the hostnames used by the STS server, LDAP server, and the DDF user connecting to CAS).
2. Open the `$(TOMCAT)/webapps/cas/WEB-INF/cas.properties` file and modify the `cas ldap.host`, `cas ldap.port`, `cas ldap.user.dn`, and `cas ldap.password` fields with your environment's LDAP information.

STS

The Security Token Service, unlike the LDAP, cannot currently be installed on a kernel distribution of DDF. To run a STS-only DDF installation, you can uninstall the catalog components that are not being used in order to increase performance. A list of unneeded components can be found on the [STS page](#).

1. In the unzipped DDF distribution folder, open `/etc/org.ops4j.pax.web.cfg` and find the line

```
org.ops4j.pax.web.ssl.keystore=etc/keystores/serverKeystore.jks
```

and change it to:

```
org.ops4j.pax.web.ssl.keystore=etc/keystores/stsKeystore.jks
```

Update the password fields to the ones your keystore uses, as well.

2. Verify that the `stsKestores.jks` file in `/etc/keystores` trusts the hostnames used in your environment (the hostnames of LDAP, CAS, and any DDF users that make use of this STS server).
3. Start the distribution.
4. Enter the following commands to install the features used by the STS server.

```
features:install security-sts-server
features:install security-cas-tokenvalidator
```

5. Open up the DDF web console as an administrator (default is admin/admin) and navigate to the Configuration tab.
6. Open up the *Security STS LDAP Login* configuration, and make sure the LDAP URL, LDAP Bind User DN, and LDAP Bind User Password fields match your LDAP server's information (the default DDF LDAP username/password is `cn=user / secret`. In a production environment, the username and password should be changed in the LDAP data file). Hit save.
7. Open up the *Security STS LDAP and Roles Claims Handler* configuration and populate the same url, user, and password fields with your LDAP server information. Hit save.
8. Open up the *Security STS CAS Token Validator* configuration. Under CAS Server URL, put the URL to your CAS server. Hit save.
9. Open up *Platform Global Configuration*. Change the protocol to https. Populate the host/port information with the STS server's host/port (for STS, the default port is 8993). Update the Trust Store and Key Store location/password fields with your environment's .jks files. Hit save.

All of the authentication components should be running and configured at this point. Now all that remains is configuring a DDF instance so it will go through this authentication scheme.

Configuring DDF instances to use the authentication scheme

Once everything is up and running, hooking up an existing DDF instance to the authentication scheme is just a matter of setting a few configuration properties.

1. Make sure the `$(DISTRIBUTION)/etc/keystores` folder is updated with the correct keystores for your operating environment.

2. Start the distribution.
3. Enter the following commands to install the CAS features

```
features:install security-cas-client
features:install security-cas-cxfServletfilter
```

4. Open up the *Security CAS Client* configuration. Under Server Name, put the URL to the STS server. Under CAS Server URL, put the URL to your CAS server. Hit save.
5. Open up *Platform Global Configuration*. Change the protocol to https and populate the host/port information with the DDF instance's host/port.
6. Update the Trust Store and Key Store location/password fields with your environment's .jks files. Hit save.
7. Open up the *Security STS Client* configuration. Make sure the host/port information in the STS Address field points to your STS server.
8. Change Signature Username to "client".
9. Change Encryption Username to "tokenissuer".
10. Change STS Token Username to "client". Hit save.

The DDF should now hit the CAS/STS/LDAP servers whenever it attempts to authenticate a user on login.

Installing DDF

Prerequisites

1. Supported platforms: *NIX - Unix/Linux/OSX, Solaris, Windows
2. **JDK 6 or 7** installed (<http://www.oracle.com/technetwork/java/javase/downloads/index.html>)

Set the JAVA_HOME environment variable to the location where the JDK is installed.

Example on how to set JAVA_HOME on *NIX

```
JAVA_HOME=/usr/java/jdk1.6.0
export JAVA_HOME
```

Example on how to set JAVA_HOME on Windows

```
set JAVA_HOME=C:\Program Files\Java\jdk1.6.0
```

*NIX

Unlink /usr/bin/java if it is already linked to a previous version of the JRE:

```
unlink /usr/bin/java
```

Verify the JAVA_HOME was set correctly

*NIX

```
echo $JAVA_HOME
```

Windows

```
echo %JAVA_HOME%
```

3. DDF installation zip file

4. A web browser
5. For Linux systems increase the file descriptor limit

Edit `/etc/sysctl.conf` and add or modify the following.

```
fs.file-max = 6815744
```

Restart

For the change to take affect a restart is required.

Restart command

```
init 6
```

The Administration Web Console is not compatible with Internet Explorer

Installation Guide

The *NIX commands listed next to the steps were performed on a default installation of Red Hat Enterprise Linux 5.4. Permission maintenance is mentioned in this article, but all files should be owned by the running user regardless of platform.

Although DDF can be installed by any user it is recommended for security reasons to have a non-root user execute the DDF installation.

Install

1. After the [prerequisites](#) have been met, (as a root user if for *NIX) change the current directory to the desired install location. This will be referred to as `<INSTALL_DIRECTORY>`.

*NIX Tip

It is recommended that the root user create a new install directory that can be owned by a non-root user (e.g. ddf-user). The non-root user (e.g. ddf-user) can now be used for the remaining installation instructions.

Example: create a directory and switch user on *NIX

```
mkdir new_installation
chown ddf-user:ddf-group new_installation

su - ddf-user
```

2. Change the current directory to location of zip file (ddf-x.y.zip).

Example in *NIX where the zip file could be located

```
cd /home/user/cdrom
```

Windows (Example assumes DDF has been downloaded to the D drive)

```
cd D:\
```

3. Copy ddf-X.Y.zip to <INSTALL_DIRECTORY>

*NIX

```
cp ddf-X.Y.zip <INSTALL_DIRECTORY>
```

Windows

```
copy ddf-X.Y.zip <INSTALL_DIRECTORY>
```

4. Change the current directory to the desired install location

*NIX or Windows

```
cd <INSTALL_DIRECTORY>
```

5. The ddf zip should now be located within the <INSTALL_DIRECTORY>. Unzip ddf-X.Y.zip.

*NIX

```
unzip ddf-X.Y.zip
```

Example of using Java to unzip in Windows

```
"C:\Program Files\Java\jdk1.6.0\bin\jar.exe" xf ddf-X.Y.zip
```

6. Run ddf by using the appropriate script.

*NIX

```
<INSTALL_DIRECTORY>/ddf-X.Y/bin/ddf
```

Windows

```
<INSTALL_DIRECTORY>/ddf-X.Y/bin/ddf.bat
```

7. Wait for the console prompt to appear.

Command prompt when initially loaded

```
ddf@local>
```

The distribution takes a few moments to load up depending on the hardware configuration. Execute the following command at the command prompt:

```
ddf@local>list
```

Proceed to Configuration.

Configuration

1. Open a compatible web browser and log in to the Administration Web Console (<http://localhost:8181/system/console>) with username: admin.
2. Select the *Configuration* tab in the Administration Web Console.
 - a. Click on the item labeled "Platform Global Configuration".
 - i. Enter the IP address or hostname of the installation in the "Host" field.
 - ii. Enter 8181 for the "Port" field. The port can be changed later if desired.
 - iii. Enter the site name (i.e., the name DDF will return in federation and web service responses).
 - iv. Click Save.
 - b. Click on the item labeled "Catalog Sorted Federation Strategy".
 - i. Enter the maximum query offset number in the "Maximum start index" field or just keep the default setting. Refer to [Standalone Framework](#) for additional info.
 - ii. Click Save.

Verification

At this point, DDF should be up and running with a Solr Catalog Provider. New features (endpoints, services, and sites) can be added as needed.

Verification can be achieved by checking that all of the DDF bundles are in an *Active* state (excluding fragment bundles which remain in a *Resolved* state).

The following command can display the status of all the DDF bundles:

```
ddf@local>list | grep -i ddf
```

If displayed, the **DDF :: Distribution :: Web Console** entry should be in the **Resolved** state. This is expected. The DDF Distribution Web Console bundle fragment. Bundle fragments are distinguished from other bundles in the command line console list by a new line under the bundle name "Hosts," followed by a bundle number. Bundle fragments remain in the **Resolved** state and can never move to the **Active** state.

What a Bundle Fragment looks like in the Command Line Console

```
[ 261] [Resolved] [          ] [          ] [ 80] DDF :: Distribution ::  
Console (2.2.0)  
Hosts: 76
```

For a complete list of installed features/bundles see the [DDF Included Features](#) document.

Troubleshooting

Exception Starting DDF

Problem: *Following exception is thrown starting DDF on a Windows machine (x86)*

Using an unsupported terminal: java.lang.NoClassDefFoundError: Could not initialize class org.fusesource.jansi.internal.Kernel32

Solution: *Install missing Windows libraries.*

Some Windows platforms are missing libraries needed by DDF. These libraries are provided by the [Microsoft Visual C++ 2008 Redistributable Package x64](#).

Starting and Stopping

Starting

*NIX

Run the following script from a command shell to start the distribution and open a local console

```
DDF_INSTALL/bin/ddf
```

Windows

Run the following script from a console window to start the distribution and open a local console

```
DDF_INSTALL/bin/ddf.bat
```

Stopping

There are two options:

- Calling shutdown from the console

Shuts down with a prompt

```
ddf@local>shutdown
```

Force Shutdown without prompt

```
ddf@local>shutdown -f
```

- Running the stop script

*NIX

```
DDF_INSTALL/bin/stop
```

Windows

```
DDF_INSTALL/bin/stop.bat
```

Shutdown

Do not shutdown by closing the window (Windows, Unix) or using the `kill -9 <pid>` command (Unix). This prevents a clean shutdown and can cause significant problems when DDF is restarted. Always use the `shutdown` command or the `Ctrl-D` keystroke (Windows) from the command line console.

Automatic Start on System Boot

Because DDF is built on top of Apache Karaf, DDF can use the Karaf Wrapper to enable automatic startup and shutdown.

1. Create the Karaf wrapper

Within the DDF console

```
ddf@local> features:install wrapper
ddf@local> wrapper:install -s AUTO_START -n ddf -d ddf -D "DDF
Service"
```

2. (Windows users skip to next step) (All *NIX) If DDF was installed to run as a non-root user (recommended) edit `DDF_INSTALL/bin/ddf-service`

Change:

DDF_INSTALL/bin/ddf-service

```
#RUN_AS_USER=
```

to:

DDF_INSTALL/bin/ddf-service

```
RUN_AS_USER=<ddf-user>
```

3. Set the memory in the wrapper config to match with DDF default memory setting
 - a) Add the setting for PermGen space under the JVM Parameters section
 - b) Update Heap space to 2048

DDF_INSTALL/etc/ddf-wrapper.conf

```
#Add the following:
wrapper.java.additional.9=-XX:PermSize=128m
wrapper.java.additional.10=-XX:MaxPermSize=512m
wrapper.java.additional.11=-Dderby.system.home="..\data\derby"
wrapper.java.additional.12=-Dderby.storage.fileSyncTransactionLog=true
wrapper.java.additional.13=-Dcom.sun.management.jmxremote
wrapper.java.additional.14=-Dfile.encoding=UTF8
wrapper.java.additional.15=-Dddf.home=%DDF_HOME%

#Update the following:
wrapper.java.maxmemory=2048
```

4. Set DDF_HOME property

DDF_INSTALL/etc/ddf-wrapper.conf

```
set.default.DDF_HOME="%KARAF_HOME%"
```

5. Install the wrapper startup/shutdown scripts

Windows

Run the following command in a console window. The command must be run with elevated permissions

```
DDF_INSTALL/bin/ddf-service.bat install
```

Startup and Shutdown settings can then be managed through the Services MMC Start -> Control Panel -> Administrative Tools -> Services

Redhat

```
root@localhost# ln -s DDF_INSTALL/bin/ddf-service /etc/init.d/
root@localhost# chkconfig ddf-service --add
root@localhost# chkconfig ddf-service on
```

Ubuntu

```
root@localhost# ln -s DDF_INSTALL/bin/ddf-service /etc/init.d/
root@localhost# update-rc.d -f ddf-service defaults
```

Solaris

```
root@localhost# ln -s DDF_INSTALL/bin/ddf-service /etc/init.d/  
root@localhost# ln -s /etc/init.d/ddf-service  
/etc/rc0.d/K20ddf-service  
root@localhost# ln -s /etc/init.d/ddf-service  
/etc/rc1.d/K20ddf-service  
root@localhost# ln -s /etc/init.d/ddf-service  
/etc/rc2.d/K20ddf-service  
root@localhost# ln -s /etc/init.d/ddf-service  
/etc/rc3.d/S20ddf-service
```

While not necessary, information on how to convert the System V init scripts to the Solaris System Management Facility can be found at <http://www.oracle.com/technetwork/articles/servers-storage-admin/scripts-to-smf-1641705.html>

Solaris specific modification

Due to a slight difference between the Linux and Solaris implementation of the "ps" command, the ddf-service script needs to be modified

1. Locate the following line in DDF_INSTALL/bin/ddf-service

Solaris DDF_INSTALL/bin/ddf-service

```
pidtest=`$PSEXEC -p $pid -o command | grep $WRAPPER_CMD | tail -1`
```

2. And modify it by truncating the word "command" to "comm"

Solaris DDF_Install/bin/ddf-service

```
pidtest=`$PSEXEC -p $pid -o comm | grep $WRAPPER_CMD | tail -1`
```

Karaf Documentation

Because DDF is built on Apache Karaf, more information on operating DDF can be found in the Karaf documentation at <http://karaf.apache.org/index/documentation.html>.

Troubleshooting

Exception Starting DDF

Problem: *Following exception is thrown starting DDF on a Windows machine (x86)*

Using an unsupported terminal: java.lang.NoClassDefFoundError: Could not initialize class org.fusesource.jansi.internal.Kernel32

Solution: *Install missing Windows libraries.*

Some Windows platforms are missing libraries needed by DDF. These libraries are provided by the [Microsoft Visual C++ 2008 Redistributable Package x64](#).

Using Console Commands

- [Command Line Shell Console](#)
 - [Accessing the System Console](#)
 - [Example Commands](#)

- [Viewing Bundle Status](#)
- [Viewing Installed Features](#)

Command Line Shell Console

When starting up the distribution, a user should notice a powerful command line console. This text console can be used to manage services, install new features and applications, or manage the state of the system.

Accessing the System Console

The Command Line Shell Console is the console that is available to the user when user starts the distribution manually. It also can be accessed from the Web Console through the [Gogo](#) tab or by using the `bin/client.bat` or `bin/client.sh` scripts. For more information on how to use the `client` scripts or how to remote into the the shell console, see [Using Remote Instances](#).

Example Commands

Viewing Bundle Status

Call `osgi:list` on the console to view the status of the bundles loaded in the distribution

Viewing Installed Features

Execute `features:list` to view the features installed in the distribution.

The majority of functionality and information available on the Web Console is also available on the Command Line Shell Console.

Catalog Commands

Catalog Commands

Title	Namespace	Description
DDF:: Catalog :: Core :: Commands	catalog	The Catalog Shell Commands are meant to be used with any <code>CatalogProvider</code> implementations. They provide general useful queries and functions against the Catalog API that can be used for debugging, printing, or scripting.

Most of commands can bypass the Catalog Framework and interact directly with the Catalog Provider if given the `-p` option if available. Hence no pre/post plugins are executed and no message validation is done if the provider (`-p`) option is used.

Catalog Commands

```
catalog:describe    catalog:dump        catalog:envlist
catalog:ingest      catalog:inspect
catalog:latest      catalog:range       catalog:remove
catalog:removeall   catalog:search
catalog:spatial
```

Command Descriptions

Command	Description
describe	Provides a basic description of the Catalog implementation.

<code>dump</code>	Exports Metacards from the local Catalog. Does not remove them.
<code>envlist</code>	Provides a list of environment variables.
<code>ingest</code>	Ingests data files into the Catalog.
<code>inspect</code>	Provides the various fields of a Metacard for inspection.
<code>latest</code>	Retrieves the latest records from the Catalog based on <code>Metacard.MODIFIED date</code> .
<code>range</code>	Searches by the given range arguments (exclusively).
<code>remove</code>	Deletes a record from the local Catalog.
<code>removeall</code>	Attempts to delete all records from the local Catalog.
<code>search</code>	Searches records in the local Catalog.
<code>spatial</code>	Searches spatially the local Catalog.

Listing Available System Console Commands

To get a list of commands, type in the namespace of the desired extension and press `<tab>`.

For example, type in: `catalog`, then `<tab>`

Getting Help for a System Console Command

For details on any command type **help** and then the command. For example, `help search`

Example Help

```
ddf@local>help search
DESCRIPTION
    catalog:search
    Searches records in the catalog provider.
SYNTAX
    catalog:search [options] SEARCH_PHRASE [NUMBER_OF_ITEMS]
ARGUMENTS
    SEARCH_PHRASE
        Phrase to query the catalog provider.
    NUMBER_OF_ITEMS
        Number of maximum records to display.
        (defaults to -1)
OPTIONS
    --help
        Display this help message
    case-sensitive, -c
        Makes the search case sensitive
    -p, -provider
        Interacts with the provider directly instead of the
framework.
```

The *help* command provides a description of the provided command, along with the syntax in how to use it, arguments it accepts, and available options.

Known Command Issues

Description
<p><i>Ingest more than 200,000 data files stored NFS shares may cause Java Heap Space error . Linux only issue.</i></p> <p>This is an NFS bug where it creates duplicate entries for some files when doing a file list. Depend on the OS, some Linux machines can handle the bug better and able get a list of files but get an incorrect number of files. Others would have a Java Heap Space error because there are too many file to list.</p>
<p><i>Ingest millions of complicated data into Solr can cause Java Heap Space error.</i></p> <p>Complicated data has spatial types and large text.</p>
<p><i>Ingest serialized data file with scientific notation in WKT string causes RuntimeException</i></p> <p>WKT string with scientific notation such as POINT (-34.8932113039107 -4.77974239601E-5) won't ingest. This happens with serialized data format only.</p>

Command Scheduler

Description

Command Scheduler is a capability exposed through the [Web Admin Console](#) that allows administrators to schedule Command Line Shell Commands to be run at specified intervals.

Usage

The Command Scheduler allows administrators to schedule [Command Line Shell Commands](#) to be run in a "platform-independent" method. For instance, if an administrator wanted to use the Catalog commands to export all records of a Catalog to a directory, the administrator could write a cron job or a scheduled task to remote into the container and execute the command. Writing these types of scripts are specific to the administrator's operating system and also requires extra logic for error handling if the container is up. The administrator instead could simply create a *Command Schedule*, which currently requires only two fields. The Command Scheduler only runs when the container is running so there would be no need to verify if the container was up. In addition, when the container is restarted, the commands are rescheduled and executed again.

Scheduling a Command

1. Navigate to the [Web Admin Console](#)
2. Click on the Configuration tab.
3. Find the *Platform Command Scheduler* configuration and click on the title or the "+" button. This will create a new configuration.
4. Type the command or commands to be executed in the **Command** text field. Commands can be separated by a semicolon and will be executed in order from left to right.
5. Type in a positive integer for the **Interval In Seconds** field.
6. Click Save. Once the Save button has been clicked, the command will be executed immediately. It's next scheduled execution happens after the amount of seconds specified in **Interval In Seconds** and repeats indefinitely until the container is shutdown or the Scheduled Command is deleted.

Scheduled Commands can be update and deleted. To delete, click on the trash bin button to the right of the page. To update, modify the fields and click Save.

Updating a Scheduled Command

1. Navigate to the [Web Admin Console](#)
2. Click on the Configuration tab.
3. Under the *Platform Command Scheduler* configuration are all the scheduled commands. Scheduled commands have the following syntax `ddf.platform.scheduler.Command.{GUID}` such as `ddf.platform.scheduler.Command.4d60c917-003a-42e8-9367-1da0f822ca6e`.
4. Find the desired configuration to modify and update either the **Command** text field or the **Interval In Seconds** field or both.
5. Click Save. Once the Save button has been clicked, the command will be executed immediately. It's next scheduled execution happens after the amount of seconds specified in **Interval In Seconds** and repeats indefinitely until the container is shutdown or the Scheduled Command is deleted.

Command Output

Commands that normally write out to the console instead will write out to the distribution's log. For instance if an `echo Hello World` command was to run every five seconds, the log would look like as follows:

Sample Command Output in the log

```
16:01:32,582 | INFO | heduler_Worker-1 | ddf.platform.scheduler.CommandJob
68 | platform-scheduler | Executing command [echo Hello World]
16:01:32,583 | INFO | heduler_Worker-1 | ddf.platform.scheduler.CommandJob
70 | platform-scheduler | Execution Output: Hello World
16:01:37,581 | INFO | heduler_Worker-4 | ddf.platform.scheduler.CommandJob
68 | platform-scheduler | Executing command [echo Hello World]
16:01:37,582 | INFO | heduler_Worker-4 | ddf.platform.scheduler.CommandJob
70 | platform-scheduler | Execution Output: Hello World
```

In other words, administrators can see the status of a run within the log as long as `INFO` was set as the status level.

Subscriptions Commands

Subscriptions Commands

Title	Namespace	Description
DDF :: Catalog :: Core :: PubSub Commands	subscriptions	The DDF PubSub Shell Commands provide functions to list the registered subscriptions in DDF and to delete subscriptions.

The Subscriptions Commands are installed by default since they are in a bundle included in the `catalog-core` feature.

Subscriptions Commands

```
ddf@local>subscriptions:
subscriptions:delete    subscriptions:list
```

Command Descriptions

Command	Description
<code>delete</code>	Deletes the subscription(s) specified by the search phrase or LDAP filter.
<code>list</code>	List the subscription(s) specified by the search phrase or LDAP filter.

Listing Available System Console Commands

To get a list of commands, type in the namespace of the desired extension and press `<tab>`.

For example, type in: `subscriptions`, then `<tab>`

Getting Help for a System Console Command

For details on any command type **help** and then the subscriptions command. For example, `help subscriptions:list`

Example Help

```
ddf@local>help subscriptions:list
DESCRIPTION
    subscriptions:list
    Allows users to view registered subscriptions.
SYNTAX
    subscriptions:list [options] [search phrase or LDAP filter]
ARGUMENTS
    search phrase or LDAP filter
        Subscription ID to search for. Wildcard characters (*) can
        be used in the ID, e.g., my*name or *123. If an id is not provided, then
        all of the subscriptions are displayed.
OPTIONS
    filter, -f
        Allows user to specify any type of LDAP filter rather than
        searching on single subscription ID.
        You should enclose the LDAP filter in quotes since it will
        often have special characters in it.
        An example LDAP filter would be:
        (& (subscription-id=my*) (subscription-id=*169*))
        which searches for all subscriptions starting with "my" and
        having 169 in the ID, which can be thought of as part of an IP address.
        An example of the entire quote command would be:
        subscriptions:list -f "(& (subscription-id=my*)
        (subscription-id=*169*))"
    --help
        Display this help message
```

The *help* command provides a description of the provided command, along with the syntax in how to use it, arguments it accepts, and available options.

subscriptions:list Command Usage Examples

Note that no arguments are required for the *subscriptions:list* command. If no argument is provided, then all subscriptions will be listed. A count of the subscriptions found matching the list command's search phrase (or LDAP filter) is displayed first, then each subscription's ID.

Listing all subscriptions

```
ddf@local>subscriptions:list

Total subscriptions found: 3

Subscription ID
my.contextual.id.v20|http://172.18.14.169:8088/mockCatalogEventConsumerBin
ding?WSDL
my.contextual.id.v30|http://172.18.14.169:8088/mockEventConsumerBinding?WS
DL
my.contextual.id.json|http://172.18.14.169:8088/services/json-ddms/local/e
vent/notification
```

Listing a specific subscription by ID

```
ddf@local>subscriptions:list  
"my.contextual.id.v20|http://172.18.14.169:8088/mockCatalogEventConsumerBinding?WSDL"  
  
Total subscriptions found: 1  
  
Subscription ID  
my.contextual.id.v20|http://172.18.14.169:8088/mockCatalogEventConsumerBinding?WSDL
```

It is recommended to always quote the search phrase (or LDAP filter) argument to the command so that any special characters will be properly processed.

Listing subscriptions using wildcards

```
ddf@local>subscriptions:list "my*"  
  
Total subscriptions found: 3  
  
Subscription ID  
my.contextual.id.v20|http://172.18.14.169:8088/mockCatalogEventConsumerBinding?WSDL  
my.contextual.id.v30|http://172.18.14.169:8088/mockEventConsumerBinding?WSDL  
my.contextual.id.json|http://172.18.14.169:8088/services/json-ddms/local/event/notification  
  
ddf@local>subscriptions:list "*json*"  
  
Total subscriptions found: 1  
  
Subscription ID  
my.contextual.id.json|http://172.18.14.169:8088/services/json-ddms/local/event/notification  
  
ddf@local>subscriptions:list "*WSDL"  
  
Total subscriptions found: 2  
  
Subscription ID  
my.contextual.id.v20|http://172.18.14.169:8088/mockCatalogEventConsumerBinding?WSDL  
my.contextual.id.v30|http://172.18.14.169:8088/mockEventConsumerBinding?WSDL
```

Listing subscriptions using an LDAP filter

The example below illustrates searching for any subscription that has "json" or "v20" anywhere in its subscription ID.

```
ddf@local>subscriptions:list -f "(|(subscription-id=*json*)
(subscription-id=*v20*))"

Total subscriptions found: 2

Subscription ID
my.contextual.id.v20|http://172.18.14.169:8088/mockCatalogEventConsumerBin
ding?WSDL
my.contextual.id.json|http://172.18.14.169:8088/services/json-ddms/local/e
vent/notification
```

The example below illustrates searching for any subscription that has "json" and "172.18.14.169" in its subscription ID. This could be a handy way of finding all subscriptions for a specific site.

```
ddf@local>subscriptions:list -f "(&(subscription-id=*json*)
(subscription-id=*172.18.14.169*))"

Total subscriptions found: 1

Subscription ID
my.contextual.id.json|http://172.18.14.169:8088/services/json-ddms/local/e
vent/notification
```

subscriptions:delete Command Usage Example

The arguments for the *subscriptions:delete* command are the same as for the *list* command, except that a search phrase or LDAP filter must be specified. If one of these is not specified an error will be displayed.

When the *delete* command is executed it will display each subscription ID it is deleting. If a subscription matches the search phrase but cannot be deleted, a message in red will be displayed with the ID. After all matching subscriptions are processed, a summary line is displayed indicating how many subscriptions were deleted out of how many matching subscriptions were found.

Deleting a specific subscription by its exact ID

```
ddf@local>subscriptions:delete
"my.contextual.id.json|http://172.18.14.169:8088/services/json-ddms/local/
event/notification"

Deleted subscription for ID =
my.contextual.id.json|http://172.18.14.169:8088/services/json-ddms/local/e
vent/notification

Deleted 1 subscriptions out of 1 subscriptions found.
```

Deleting subscriptions using wildcards


```
ddf@local>subscriptions:delete "my*"
```

```
Deleted subscription for ID =  
my.contextual.id.v20|http://172.18.14.169:8088/mockCatalogEventConsumerBin  
ding?WSDL
```

```
Deleted subscription for ID =  
my.contextual.id.v30|http://172.18.14.169:8088/mockEventConsumerBinding?WS  
DL
```

```
Deleted 2 subscriptions out of 2 subscriptions found.
```

```
ddf@local>subscriptions:delete "*json*"
```

```
Deleted subscription for ID =  
my.contextual.id.json|http://172.18.14.169:8088/services/json-ddms/local/e  
vent/notification
```

```
Deleted 1 subscriptions out of 1 subscriptions found.
```

Deleting all subscriptions

```
ddf@local>subscriptions:delete *
```

```
Deleted subscription for ID =  
my.contextual.id.v30|http://172.18.14.169:8088/mockEventConsumerBinding?WS  
DL
```

```
Deleted subscription for ID =  
my.contextual.id.v20|http://172.18.14.169:8088/mockCatalogEventConsumerBin  
ding?WSDL
```

```
Deleted subscription for ID =  
my.contextual.id.json|http://172.18.14.169:8088/services/json-ddms/local/e  
vent/notification
```

```
Deleted 3 subscriptions out of 3 subscriptions found.
```

Deleting subscriptions using an LDAP filter

```
ddf@local>subscriptions:delete -f "(&(subscription-id=*WSDL)
(subscription-id=*172.18.14.169*))"

Deleted subscription for ID =
my.contextual.id.v20|http://172.18.14.169:8088/mockCatalogEventConsumerBin
ding?WSDL
Deleted subscription for ID =
my.contextual.id.v30|http://172.18.14.169:8088/mockEventConsumerBinding?WS
DL

Deleted 2 subscriptions out of 2 subscriptions found.
```

Configuration

DDF can be configured in several ways:

- [Configuration via the Web Console](#)
- [Configuration via the System Console](#)
- [Configuration via Configuration \(.cfg\) Files](#)

Configuration via the Web Console

- [Accessing the Web Console](#)
- [Configuring DDF Platform Global Settings](#)
- [Managing Features](#)
- [Known Issues](#)
- [Additional Information](#)

Accessing the Web Console

Open the web administration console.

- <http://localhost:8181/system/console>
- Enter Username and Password

The default username/password is admin/admin. To change this refer to the [Hardening](#) section.

Configuring DDF Platform Global Settings

1. Open the web administration console.
 - <http://localhost:8181/system/console>
 - Enter Username and Password

The default username/password is admin/admin. To change this refer to the [Hardening](#) section.

2. Select the *Configuration* tab
 3. Select the *Platform Global Configuration* bundle
 4. Enter the DDF configuration values per the Configurable Properties table below.
 5. Select the "Save" Button
-

Configurable Properties

Title	Property	Type	Description	Default Value	Required
Protocol	protocol	String	Default protocol that should be used to connect to this machine.	http	yes
Host	host	String	The host name or IP address of the machine that DDF is running on. Do not enter localhost.		yes
Port	port	String	The port that DDF is running on.		yes
Trust Store	trustStore	String	The trust store used for outgoing SSL connections. Path is relative to ddf.home.	etc/keystores/clientTruststore.jks	no
Trust Store Password	trustStorePassword	String	The password associated with the trust store.	changeit (encrypted)	no
Key Store	keyStore	String	The key store used for outgoing SSL connections. Path is relative to karaf.home.	etc/keystores/clientKeystore.jks	no
Key Store Password	keyStorePassword	String	The password associated with the key store.	changeit (encrypted)	no
Site Name	id	String	The site name for DDF	ddf.distribution	yes
Version	version	String	The version of DDF that is running. This value should not be changed from the factory default.	2.2.1	yes
Organization	organization	String	The organization responsible for this installation of DDF	Codice Foundation	yes

Managing Features

DDF includes many components, packaged as features, that can be installed and/or uninstalled without restarting the system. Features are collections of OSGi bundles, configuration data, and/or other features. For more information on the features that come with DDF, including a list of the ones included, consult the [DDF Included Features](#) page in the Software Version Description Document (SVDD).

Transitive Dependencies

Features may have dependencies on other features and will auto-install them as needed.

Installing Features via the Web Console

1. Open the web administration console.
 - <http://localhost:8181/system/console>
 - Enter Username and Password

The default username/password is admin/admin. To change this refer to the [Hardening](#) section.

2. Select the *Features* tab.

Admin	Bundles	Components	Configuration	Configuration Status	Deployment Packages	Events	Features
-------	---------	------------	---------------	----------------------	---------------------	--------	----------

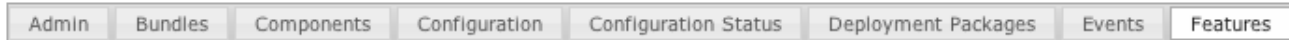
3. Click on the 'Play' arrow under the Actions column for the feature that should be installed.
4. Wait for the Status to change from *Uninstalled* to *Installed*
5. Optional: Check Bundle Status
 - a. Select the *Bundles* tab
 - b. Scroll down to the associated bundles and verify they were installed with a status of *Active*

Uninstalling Features

1. Open the web administration console.
 - <http://localhost:8181/system/console>
 - Enter Username and Password

The default username/password is admin/admin. To change this refer to the [Hardening](#) section.

2. Select the *Features* tab.



3. Click on the 'Eject' icon under the *Actions* column for the feature that should be uninstalled.
4. Wait for the Status to change from *Installed* to *Uninstalled*
5. Optional: Check Bundle Status
 - a. Select the *Bundles* tab
 - b. Verify the absence of the associated bundle(s)

Adding Feature Repositories

1. Open the web administration console.
 - <http://localhost:8181/system/console>
 - Enter Username and Password

The default username/password is admin/admin. To change this refer to the [Hardening](#) section.

2. Select the *Features* tab.
3. Enter the URL to the feature repository (see below) to be added and click the Add URL button.
5. The new feature repository should be added to the list of Feature Repositories above where you entered the URL.

There are several ways a new feature repository can be discovered based on the URL entered.

The URL can contain the fully qualified path to the feature repository, e.g., `mvn:https://tools.codice.org/artifacts/content/groups/public/ddf-standard/2.2.0/xml/features`. This URL can include the username and password credentials if necessary. e.g., `mvn:https://user/password@tools.codice.org/artifacts/content/groups/public/ddf-standard/2.2.0/xml/features`. Note that the password will be in clear text.

The URL can just be the mvn URL, e.g., `mvn:ddf.features/ddf-standard/2.2/xml/features` and the feature repositories configured for DDF will be searched.

Repositories to be searched by DDF can be configured in one of several ways:

- By setting the `org.ops4j.pax.url.mvn.repositories` property in the `<DDF_INSTALL_DIR>/etc/org.ops4j.pax.url.mvn.cfg` file (most common)
- **or** by setting the `org.ops4j.pax.url.mvn.settings` property in the `<DDF_INSTALL_DIR>/etc/org.ops4j.pax.url.mvn.cfg` file to the `maven.settings.xml` file that specifies the repository(ies) to be searched
Note that a `settings.xml` template file for this configuration is provided in `<DDF_INSTALL_DIR>/etc/templates/settings.xml`
- **or** by having a `maven.settings.xml` file in the `<USER_HOME_DIR>/m2` directory of the user running DDF that specifies the repository(ies) to be searched (this would be typical for a developer)

The simplest approach is to specify the fully qualified URL to the feature repository.

Known Issues

Blank Web Console

DDF uses Pax Web as part of its HTTP support. Modifying the Pax Web runtime configuration in the web console may cause the web console to freeze.

Solution

Use the configuration instructions according to the [hardening instructions](#).

Additional Information

For more information on the Web Console see the following link: <http://felix.apache.org/site/apache-felix-web-console.html>

Configuration via the System Console

- [Managing Features](#)
 - [Installing Features](#)
 - [Uninstalling Features](#)

Using the System Console is described in the [Using Console Commands](#) section.

Managing Features

DDF includes many components, packaged as features, that can be installed and/or uninstalled without restarting the system. Features are collections of OSGi bundles, configuration data, and/or other features. For more information on the features that come with DDF, including a list of the ones included, consult the [DDF Included Features](#) page in the Software Version Description Document (SVDD).

Transitive Dependencies

Features may have dependencies on other features and will auto-install them as needed.

Installing Features

1. Determine which feature to install by looking at the available features on DDF.

```
ddf@local>features:list
```

The console should output a list of all features available (installed and uninstalled). A snippet of the list output is shown below (the versions may differ based on the version of DDF being run):

State	Version	Name
Repository	Description	
[installed]	[2.0.1] ddf-core ddf-2.1.0
[uninstalled]	[2.0.1] ddf-sts ddf-2.1.0
[installed]	[2.0.1] ddf-security-common ddf-2.1.0
[installed]	[2.0.1] ddf-resource-impl ddf-2.1.0
[uninstalled]	[2.0.1] ddf-source-dummy ddf-2.1.0

2. Install the desired feature.

```
ddf@local>features:install ddf-source-dummy
```

3. Check the feature list to verify the feature was installed.

```
ddf@local>features:list
```

State	Version	Name	
Repository	Description		
[installed]	[2.0.1] ddf-core	ddf-2.1.0
[uninstalled]	[2.0.1] ddf-sts	ddf-2.1.0
[installed]	[2.0.1] ddf-security-common	ddf-2.1.0
[installed]	[2.0.1] ddf-resource-impl	ddf-2.1.0
[installed]	[2.0.1] ddf-source-dummy	ddf-2.1.0

4. Check the bundle status to verify the service is started.

```
ddf@local>list
```

The console output should show an entry similar to the following:

```
[ 117] [Active      ] [           ] [Started] [ 75] DDF :: Catalog
:: Source :: Dummy (<version>)
```

Uninstalling Features

1. Check the feature list to verify the feature is installed properly.

```
ddf@local>features:list
```

State	Version	Name	
Repository	Description		
[installed]	[2.0.1] ddf-core	ddf-2.1.0
[uninstalled]	[2.0.1] ddf-sts	ddf-2.1.0
[installed]	[2.0.1] ddf-security-common	ddf-2.1.0
[installed]	[2.0.1] ddf-resource-impl	ddf-2.1.0
[installed]	[2.0.1] ddf-source-dummy	ddf-2.1.0

2. Uninstall the feature

```
ddf@local>features:uninstall ddf-source-dummy
```

Dependencies that were auto-installed by the feature are not automatically uninstalled.

3. Verify that the feature has uninstalled properly.

```
ddf@local>features:list
```

State	Version	Name	
Repository	Description		
[installed]	[2.0.1] ddf-core	ddf-2.1.0
[uninstalled]	[2.0.1] ddf-sts	ddf-2.1.0
[installed]	[2.0.1] ddf-security-common	ddf-2.1.0
[installed]	[2.0.1] ddf-resource-impl	ddf-2.1.0
[uninstalled]	[2.0.1] ddf-source-dummy	ddf-2.1.0

Configuration via Configuration (.cfg) Files

HTTP Port Configuration

Do not use the Web Administration Console to change the HTTP port. While the Web Administration Console's Pax Web Runtime offers this configuration option, it has proven to be unreliable and may crash the system

Multiple Local DDF Nodes

1. a. Make port number edits to files in the DDF install folder. Line numbers are referenced for 2.1.X releases.

File to Edit	Line Number	Original Value	Example of New Value
bin/karaf.bat	99	5005	i.e. 5006
etc/org.apache.karaf.management.cfg	27	1099	i.e. 1199
" "	32	44444	i.e. 44445
etc/org.ops4j.pax.web.cfg	9	8181	i.e. 8281
" "	22	8993	i.e. 8994

Be sure to note the port number that replaced "8181" and enter that in the Web Console under the Configuration tab for the Platform Global Configuration -> DDF Port entry. Also edit the sitename so that there are no duplicates on your local machine.

Keep in mind that only root can access ports < 1024 on Unix systems. For suggested ways to run DDF with ports < 1024 see [How do I use port 80 as a non-root user?](#)

Enabling SSL for Services

Do not use the Web Administration Console to SSL enable the DDF services. While the Web Administration Console's Pax Web Runtime offers this configuration option, it has proven to be unreliable and may crash the system.

Edit the provided configuration file <DDF_INSTALL_DIR>/etc/org.ops4j.pax.web.cfg with the settings for the desired configuration. Pax Web Configuration Settings

Property	Sample Value	Description
org.osgi.service.http.enabled	false	Set this to false to disable HTTP without SSL
org.osgi.service.http.secure.enabled	true	Set this to true to SSL enable the DDF services

org.osgi.service.http.port.secure	8993	Set this to the HTTPS port number. (Verify this port does not conflict with any other secure ports being used in the network. For example, JBoss and other application servers use port 8443 by default)
org.ops4j.pax.web.ssl.keystore.type	jks	Set this to the type of keystore (most likely jks)
org.ops4j.pax.web.ssl.keystore	/opt/ddf/keystore.jks	Set this to the fully-qualified path to the SSL keystore file
org.ops4j.pax.web.ssl.keypassword	password1	Set this to the password for the user's private key
org.ops4j.pax.web.ssl.password	password2	Set this to the password for overall keystore integrity checking

Here is an example .cfg file:

```
#####
# HTTP settings
#####

# Disable HTTP
org.osgi.service.http.enabled=false

# HTTP port number
org.osgi.service.http.port=8181


#####
# HTTPS settings
#####

# Enable HTTPS
org.osgi.service.http.secure.enabled=true

# HTTPS port number
# (Verify this port does not conflict with any other secure ports being
# used in the
# network. For example, JBoss and other application servers use port 8443
# by default)

org.osgi.service.http.port.secure=8993

# Fully-qualified path to the SSL keystore
org.ops4j.pax.web.ssl.keystore=/opt/ddf/keystore.jks

# SSL Keystore Type
org.ops4j.pax.web.ssl.keystore.type=jks

# Keystore Integrity Password
org.ops4j.pax.web.ssl.password=abc123

# Keystore Password
org.ops4j.pax.web.ssl.keypassword=abc123
```


All `.cfg` files follow a strict formatting structure in that every entry is a key=value pair. There should be no whitespace before the key, around the equals sign (=), or after the value. Otherwise, the key or value may be misinterpreted.

Also take care if `.cfg` files originated on an operating system other than the operating system DDF is currently running on. Hidden characters, e.g., `^M`, can be added during the file transfer between the operating systems. This occurs often when a DDF zip install file from a Unix operating system is transferred to a Windows operating system and installed.

Optional: Disable HTTP for the DDF services and only use HTTPS by setting the `org.osgi.service.http.enabled` property to `false`. After this, all DDF clients need to pass the appropriate certificates.

Reference

[Configuring a Java Keystore for Secure Communications](#)

Additional Pax-Web SSL configuration info: <http://team.ops4j.org/wiki/display/paxweb/SSL+Configuration>

How do I use port 80 as a non-root user?

This documentation has been reproduced from the [Jetty website](#) for easy reference.

On Unix based systems, port 80 is protected and can usually only be opened by the superuser root. As it is not desirable to run the server as root (for security reasons), the possible solutions are:

- Start Jetty as the root user, and use Jetty's `setuid` mechanism to switch to a non-root user after startup, or
- Configure the server to run as a normal user on port 8080 (or some other non protected port). Then, configure the operating system to redirect port 80 to 8080 using `ipchains`, `iptables`, `ipfw` or a similar mechanism.

The latter has traditionally been the solution, however Jetty 6.1 has added the new `setuid` feature.

If using Solaris 10, this feature may not be needed as Solaris provides a User Rights Management framework that can permit users and processes superuser-like abilities. Refer to the [Solaris documentation](#) for more information.

Using Jetty's `setuid` (and `setumask`) feature

Create a jetty config file like the following:

```
<?xml version="1.0"?>
<!DOCTYPE Configure PUBLIC "-//Mort Bay Consulting//DTD Configure//EN"
"http://jetty.mortbay.org/configure.dtd">
<Configure id="Server" class="org.mortbay.setuid.SetUIDServer">
  <Set name="umask">UMASK</Set>
  <Set name="uid">USERID</Set>
</Configure>
```

Where:

- **UMASK** is replaced with the umask setting the process needs, or optionally remove this line if the process umask setting does not need to be changed at runtime
- **USERID** is replaced with the id of the user process needs to execute as once the ports have been opened.

Hint

An existing Jetty configuration file like the above is located in the `$jetty.home/extras/setuid/etc/jetty-setuid.xml`.

Next build the `setuid` feature for the specific operating system, as it requires native libraries. Go to the `$jetty.home/extras/setuid` directory and follow the instructions in the `README.txt` file, summarized here as:

```

> mvn install

> gcc -I$JDK_HOME/include/ -I$JDK_HOME/include/linux/ \
    -shared src/main/native/org_mortbay_setuid_SetUID.c \
    -o ../../lib/ext/libsetuid.so

> cp target/jetty-setuid-6.1-SNAPSHOT.jar ../../lib/ext/
> cp etc/jetty-setuid.xml ../../etc

```

Where:

- **\$JDK_HOME** is same as \$JAVA_HOME
- **linux** should be replaced by the name of DDF's operating system.

On Solaris

Omit the `-shared` argument.

Then run jetty as the root user by switching to the userid (and setting the umask, if configured):

```

sudo java -Djava.library.path=lib/ext -jar start.jar
etc/jetty-setuid.xml etc/jetty.xml

```

Note!

The `etc/jetty-setuid.xml` file must be first in the list of config files.

Using ipchains

On some Linux systems the `ipchains` REDIRECT mechanism can be used to redirect from one port to another inside the kernel:

```

/sbin/ipchains -I input --proto TCP --dport 80 -j REDIRECT 8080

```

This basically means, "Insert into the kernel's packet filtering the following as the first rule to check on incoming packets: If the protocol is TCP and the destination port is 80, redirect the packet to port 8080." The kernel must be compiled with support for `ipchains`. (virtually all stock kernels are.) The `ipchains` command-line utility must be installed. (On RedHat the package is named "ipchains".) This command can run at any time, preferably just once since it inserts another copy of the rule every time it is executed.

Once this rule is set up, a Linux 2.2 kernel will redirect all data addressed to port 80 to a server such as Jetty running on port 8080. This includes all RedHat 6.x distros. Linux 2.4 kernels, e.g. RedHat 7.1+, have a similar `iptables` facility.

Using iptables

A command similar to the following must be added to the startup scripts or the firewall rules:

```

/sbin/iptables -t nat -I PREROUTING -p tcp --dport 80 -j REDIRECT --to-port
8080

```

The underlying model of `iptables` is different to that of `ipchains` so the forwarding normally only happens to packets originating off-box. Incoming packets must be routed to port 8080 if `iptables` is used as a local firewall.

Be careful to place rules like this early in the "input" chain. Such rules must precede any rule that would accept the packet, otherwise the redirection won't occur. As many rules as needed can be inserted if the server needs to listen on multiple ports, as for HTTPS.

Using usermod

On Solaris 10 (possibly including earlier versions) the OS allows privileges to be granted to ports binding to "normal" users:

```
usermod -K defaultpriv=basic,net_privaddr myself
```

Now the `myself` user will be able to bind to port 80.

Using xinetd

With modern Linux flavors, `inetd` has an improved command `xinetd`. There are existing Unix man pages that describes its functionality in detail.

`xinetd` can be used to redirect network traffic, requiring configuration with only a text editor.

`xinetd` is driven by text files. There are 2 ways to give `xinetd` instructions:

1. Add a new service to `etc/xinetd.conf`
2. Add a new file to the directory `etc/xinetd.d`

For both approaches the format is the same.

The following entry will redirect all inward TCP traffic on port 80 to port 8888 on the local machine. Redirection to other machines is also possible for `gimp` proxying:

```
service my_redirector
{
    type = UNLISTED
    disable = no
    socket_type = stream
    protocol = tcp
    user = root
    wait = no
    port = 80
    redirect = 127.0.0.1 8888
    log_type = FILE /tmp/somefile.log
}
```

Points to Note

- A space must be on either side of the '=' or the line is ignored.
- `type = UNLISTED` means that the name of the service does not have to be in `/etc/services`, but the port and protocol must be specified. If an existing service name, e.g., `http`, is used:

```
service http
{
    disable = no
    socket_type = stream
    user = root
    wait = no
    redirect = 127.0.0.1 8888
    log_type = FILE /tmp/somefile.log
}
```

Examine the `/etc/services` for more info on this configuration.

- Logging may present certain security problems, hence it can be omitted if necessary
- RHEL5 does not contain `xinetd` by default. Doing a `yum install xinetd` will fix this.

HTTP Port Configuration

HTTP Port Configuration

Do not use the Web Administration Console to change the HTTP port. While the Web Administration Console's Pax Web Runtime offers this configuration option, it has proven to be unreliable and may crash the system

Multiple Local DDF Nodes

1. a. Make port number edits to files in the DDF install folder. Line numbers are referenced for 2.1.X releases.

File to Edit	Line Number	Original Value	Example of New Value
bin/karaf.bat	99	5005	i.e. 5006
etc/org.apache.karaf.management.cfg	27	1099	i.e. 1199
" "	32	44444	i.e. 44445
etc/org.ops4j.pax.web.cfg	9	8181	i.e. 8281
" "	22	8993	i.e. 8994

Be sure to note the port number that replaced "8181" and enter that in the Web Console under the Configuration tab for the Platform Global Configuration -> DDF Port entry. Also edit the sitename so that there are no duplicates on your local machine.

Keep in mind that only root can access ports < 1024 on Unix systems. For suggested ways to run DDF with ports < 1024 see [How do I use port 80 as a non-root user?](#)

SSL Enable Services

Enabling SSL for Services

Do not use the Web Administration Console to SSL enable the DDF services. While the Web Administration Console's Pax Web Runtime offers this configuration option, it has proven to be unreliable and may crash the system.

Edit the provided configuration file `<DDF_INSTALL_DIR>/etc/org.ops4j.pax.web.cfg` with the settings for the desired configuration.
Pax Web Configuration Settings

Property	Sample Value	Description
org.osgi.service.http.enabled	false	Set this to false to disable HTTP without SSL
org.osgi.service.http.secure.enabled	true	Set this to true to SSL enable the DDF services
org.osgi.service.http.port.secure	8993	Set this to the HTTPS port number. (Verify this port does not conflict with any other secure ports being used in the network. For example, JBoss and other application servers use port 8443 by default)
org.ops4j.pax.web.ssl.keystore.type	jks	Set this to the type of keystore (most likely jks)
org.ops4j.pax.web.ssl.keystore	/opt/ddf/keystore.jks	Set this to the fully-qualified path to the SSL keystore file
org.ops4j.pax.web.ssl.keypassword	password1	Set this to the password for the user's private key

org.ops4j.pax.web.ssl.password	password2	Set this to the password for overall keystore integrity checking
--------------------------------	-----------	--

Here is an example .cfg file:

```
#####
# HTTP settings
#####

# Disable HTTP
org.osgi.service.http.enabled=false

# HTTP port number
org.osgi.service.http.port=8181


#####
# HTTPS settings
#####

# Enable HTTPS
org.osgi.service.http.secure.enabled=true

# HTTPS port number
# (Verify this port does not conflict with any other secure ports being
used in the
# network. For example, JBoss and other application servers use port 8443
by default)

org.osgi.service.http.port.secure=8993

# Fully-qualified path to the SSL keystore
org.ops4j.pax.web.ssl.keystore=/opt/ddf/keystore.jks

# SSL Keystore Type
org.ops4j.pax.web.ssl.keystore.type=jks

# Keystore Integrity Password
org.ops4j.pax.web.ssl.password=abc123

# Keystore Password
org.ops4j.pax.web.ssl.keypassword=abc123
```

All .cfg files follow a strict formatting structure in that every entry is a key=value pair. There should be no whitespace before the key, around the equals sign (=), or after the value. Otherwise, the key or value may be misinterpreted.

Also take care if .cfg files originated on an operating system other than the operating system DDF is currently running on. Hidden characters, e.g., ^M, can be added during the file transfer between the operating systems. This occurs often when a DDF zip install file from a Unix operating system is transferred to a Windows operating system and installed.

Optional: Disable HTTP for the DDF services and only use HTTPS by setting the `org.osgi.service.http.enabled` property to `false`. After this, all DDF clients need to pass the appropriate certificates.

Reference

Configuring a Java Keystore for Secure Communications

Additional Pax-Web SSL configuration info: <http://team.ops4j.org/wiki/display/paxweb/SSL+Configuration>

Enable HTTP Access Logging

To enable access logs for our current DDF do the following:

1. Update the file (`org.ops4j.pax.web.cfg`) located in `etc/` by adding the following text to the bottom:

```
org.ops4j.pax.web.config.file=etc/jetty.xml
```

This tells pax-web to look at the external `jetty.xml` file when performing initial jetty configuration.

2. Update the `jetty.xml` file located in `etc/` adding the following xml:

```
<Get name="handler">
  <Call name="addHandler">
    <Arg>
      <New
class="org.eclipse.jetty.server.handler.RequestLogHandler">
      <Set name="requestLog">
        <New id="RequestLogImpl"
class="org.eclipse.jetty.server.NCSARequestLog">
          <Arg><SystemProperty name="jetty.logs"
default="data/log/" />/yyyy_mm_dd.request.log</Arg>
          <Set name="retainDays">90</Set>
          <Set name="append">true</Set>
          <Set name="extended">false</Set>
          <Set name="LogTimeZone">GMT</Set>
        </New>
      </Set>
    </New>
    </Arg>
  </Call>
</Get>
```

3. Change the location of the logs to the desired location, in settings above will default to `data/log` (same place where the log is located).

The log is using NCSA format (hence the class 'NCSARequestLog'). This is the most popular format for access logs and can be parsed by many web server analytics tools. Here is a sample output:

```
127.0.0.1 - - [14/Jan/2013:16:21:24 +0000] "GET /favicon.ico HTTP/1.1"
200 0
127.0.0.1 - - [14/Jan/2013:16:21:33 +0000] "GET /services/ HTTP/1.1" 200
0
127.0.0.1 - - [14/Jan/2013:16:21:33 +0000] "GET /services/?stylesheet=1
HTTP/1.1" 200 0
127.0.0.1 - - [14/Jan/2013:16:21:33 +0000] "GET /favicon.ico HTTP/1.1"
200 0
```

Additional information:

<http://team.ops4j.org/wiki/display/paxweb/Advanced+Jetty+Configuration>

<http://wiki.eclipse.org/Jetty/Tutorial/RequestLog>

Configuring a Java Keystore for Secure Communications

The following information was sourced from <https://www.racfi.bnl.gov/terapaths/software/the-terapaths-api/example-java-client/java-client/setting-up-keystores-with-jetty-and-keytool>.

Creating a Client Keystore

This walk-through details how to use a PKCS12 certificate. This is the most popular format used when exporting from a web browser.

1. Obtain a personal ECA cert (client certificate)
 - a. To do this, open Internet Explorer > Tools > Options.
 - b. Select the Content tab.
 - c. Click Certificates
 - d. Select the Personal tab.
 - e. Select the certificate needed to export. There should be the one without a "Friendly Name" and it is **not** the "Encryption Cert").
 - f. Click Export.
 - g. Follow Certificate Export Wizard.
 - h. When a prompt requests to export the private key, select Yes.
2. Download a jetty 6.1.5 distribution from <http://dist.codehaus.org/jetty/jetty-6.1.5/jetty-6.1.5.zip>
3. Unpack the jetty distribution and place the client certificate (the one just exported) in the lib directory.
4. Navigate to the lib directory of the jetty distribution in a command console.
5. Add a *cert* to a new java keystore, replacing *cert* with the name of the PKCS12 Keystore needed to convert, and replace *clientKeystore* with the desired name of the Java Keystore:

```
java -cp jetty-6.1.5.jar org.mortbay.jetty.security.PKCS12Import
cert.p12 clientKeystore.jks
```

6. The command prompts for two passwords:
 - a. Input keystore passphrase is the passphrase that is used to protect *cert.p12*
 - b. Output keystore passphrase is the passphrase that is set for the new java keystore *clientKeystore.jks*
7. It is recommended that the private key password be the same as the keystore password due to limitations in java.
 - a. Run the following command to determine the alias name of the added current entry. It is listed after "Alias Name:"

```
keytool -list -v -keystore clientKeystore.jks
```

- b. Clone the existing key using the java keytool executable, filling in *<CurrentAlias>*, *<NewAlias>*, *clientKeystore.jks*, and *password* with the correct names.

```
keytool -keyclone -alias "<CurrentAlias>" -dest "<NewAlias>"  
-keystore clientKeystore.jks -storepass password
```

- c. When prompted for a password, use the same password used when the keystore was created.
- d. Delete the original alias

```
keytool -delete -alias "<CurrentAlias>" -keystore  
clientKeystore.jks -storepass password
```

After the keystore is successfully created, delete the jetty files used to perform the import.

Creating a Truststore

This walk-through details how to import a .cer certificate

1. Import the certificate into a java keystore as a trusted ca certificate

```
keytool -import -trustcacerts -alias "Trusted Cert" -file  
trustcert.cer -keystore truststore.jks
```

2. Enter in a keystore password when prompted.

Adding a certificate to an existing Keystore

1. Import the certificate into a java keystore as a certificate.

```
keytool -importcert -file newcert.cer -keystore clientKeystore.jks  
-alias "New Alias"
```

2. Enter in the keystore password if prompted.

Hardening

Introduction

Disabling the Web Administration Console

- Enabling SSL for Services
- Pax Web Configuration Settings

Configuring DDF Without the Web Administration Console

- Configuring Using a .cfg File Template
- Configuring Using the Command Line Console

Introduction

For more secure installations, these instructions demonstrate how to harden a DDF system.

The web administration console is not compatible with Internet Explorer 7

Disabling the Web Administration Console

To harden DDF for security purposes, disable the Web Administration Console so that users do not have access. All configuration is done via the DDF command line console and/or `.cfg` configuration files.

Open the web administration console.

1. <http://localhost:8181/system/console>
2. Default **Username/Password**: admin/admin
3. Select the *Features* tab
4. Uninstall the `webconsole-base` feature by clicking the **Install/Uninstall** button on the right

Accessing any page on the web admin console will now result in an HTTP 404 error.

To re-enable the web administration console, go to the Karaf command line console and install the `ddf-webconsole` feature.

```
features:install webconsole
```

Enabling SSL for Services

Do not use the Web Administration Console to SSL enable the DDF services. While the Web Administration Console's Pax Web Runtime offers this configuration option, it has proven to be unreliable and may crash the system.

Edit the provided configuration file `<DDF_INSTALL_DIR>/etc/org.ops4j.pax.web.cfg` with the settings for the desired configuration.

Pax Web Configuration Settings

Property	Sample Value	Description
<code>org.osgi.service.http.enabled</code>	<code>false</code>	Set this to false to disable HTTP without SSL
<code>org.osgi.service.http.secure.enabled</code>	<code>true</code>	Set this to true to SSL enable the DDF services
<code>org.osgi.service.http.port.secure</code>	<code>8993</code>	Set this to the HTTPS port number. (Verify this port does not conflict with any other secure ports being used in the network. For example, JBoss and other application servers use port 8443 by default)
<code>org.ops4j.pax.web.ssl.keystore.type</code>	<code>jks</code>	Set this to the type of keystore (most likely <code>jks</code>)
<code>org.ops4j.pax.web.ssl.keystore</code>	<code>/opt/ddf/keystore.jks</code>	Set this to the fully-qualified path to the SSL keystore file
<code>org.ops4j.pax.web.ssl.keypassword</code>	<code>password1</code>	Set this to the password for the user's private key
<code>org.ops4j.pax.web.ssl.password</code>	<code>password2</code>	Set this to the password for overall keystore integrity checking

Here is an example `.cfg` file:

```
#####
# HTTP settings
#####

# Disable HTTP
org.osgi.service.http.enabled=false

# HTTP port number
org.osgi.service.http.port=8181


#####
# HTTPS settings
#####

# Enable HTTPS
org.osgi.service.http.secure.enabled=true

# HTTPS port number
# (Verify this port does not conflict with any other secure ports being
used in the
# network. For example, JBoss and other application servers use port 8443
by default)

org.osgi.service.http.port.secure=8993

# Fully-qualified path to the SSL keystore
org.ops4j.pax.web.ssl.keystore=/opt/ddf/keystore.jks

# SSL Keystore Type
org.ops4j.pax.web.ssl.keystore.type=jks

# Keystore Integrity Password
org.ops4j.pax.web.ssl.password=abc123

# Keystore Password
org.ops4j.pax.web.ssl.keypassword=abc123
```

All `.cfg` files follow a strict formatting structure in that every entry is a key=value pair. There should be no whitespace before the key, around the equals sign (=), or after the value. Otherwise, the key or value may be misinterpreted.

Also take care if `.cfg` files originated on an operating system other than the operating system DDF is currently running on. Hidden characters, e.g., `^M`, can be added during the file transfer between the operating systems. This occurs often when a DDF zip install file from a Unix operating system is transferred to a Windows operating system and installed.

Optional: Disable HTTP for the DDF services and only use HTTPS by setting the `org.osgi.service.http.enabled` property to `false`. After this, all DDF clients need to pass the appropriate certificates.

Reference

Configuring a Java Keystore for Secure Communications

Additional Pax-Web SSL configuration info: <http://team.ops4j.org/wiki/display/paxweb/SSL+Configuration>

Unable to render {include} The included page could not be found.

Configuring DDF Without the Web Administration Console

Depending on the environment, it may be easier for integrators and administrators to configure DDF using the Web Administration Console prior to disabling it and switching to SSL. The Web Administration Console can be re-enabled for additional configuration changes.

In an environment hardened for security purposes, access to the DDF Web Administration Console will be denied. It is necessary to configure DDF (e.g., providers, Schematron rulesets, etc.) either using `.cfg` configuration files or by using the Karaf command line console. The OSGi container detects the addition, updating, or deletion of `.cfg` files in the `etc/ddf` directory.

The following sections describe how to configure each DDF item using both of these mechanisms. A template file is provided for each configurable DDF item so that it can be copied/renamed and then modified with settings.

If at a later time the Web Administration console is enabled again, all of the configuration done via `.cfg` files and/or the Karaf command line console are loaded and displayed. However, note that the name of the `.cfg` file is not used in the admin console. Rather, OSGi assigns a universally unique identifier (UUID) when the DDF item was created and displays this UUID in the console (e.g., `OpenSearchSource.112f298e-26a5-4094-befc-79728f216b9b`)

Templates included with DDF:

DDF Service	Template File Name	Factory PID	Configurable Properties (from DDF Users Guide)
DDF Catalog Framework	<code>ddf.catalog.impl.service.CatalogFrameworkImpl.cfg</code>	<code>ddf.catalog.CatalogFrameworkImpl</code>	Standard Catalog Framework

Configuring Using a .cfg File Template

Configuring a new source or feature using a config file follows a standard procedure:

- Copy/rename the provided template file in the `etc/templates` directory to the `etc` directory (consult preceding table to determine correct template).
 - Mandatory: The dash between the PID (e.g., `OpenSearchSource_site.cfg`) and the instance name (e.g., `OpenSearchSource_site.cfg`) is required. The dash is a reserved character used by OSGi to detect that an instance of a managed service factory should be created.**
 - Not required, but a good practice is to change the instance name (e.g. `federated_source`) of the file to something identifiable (`source1-ddf`)
- Edit the file copied to `etc` with the settings for the configuration (consult preceding table to determine the configurable properties).
 - This file is a Java properties file, hence the syntax is `<key> = <value>`
 - Consult the inline comments in the file for guidance on what to modify
 - The Configurable Properties tables in the Integrator's Guide for the Included Catalog Components also describe each field and its value
- The new service can now be used in the same way as if it was created using the web administration console.

Configuring Using the Command Line Console

Configuring a new source, provider, or feature using the command console follows a standard procedure. The properties and their values will change based on type of service being created, but the actual commands entered at the command line do not change.

To help illustrate the commands, an example of creating a new OpenSearch Federated Source is shown after each step.

- Create the federated source using the `config:edit` command
 - Mandatory: The dash between the PID (e.g., `OpenSearch_source`) and the instance name (e.g., `search_source`) is required. The dash is a reserved character used by OSGi to detect that an instance of a managed service factory should be created.**

```
config:edit OpenSearchSource-my_federated_source
```

2. Enter the settings for the federated source's properties. These properties can be found by looking in the template file for the specified service.

```
config:propset endpointUrl
http://ddf:8181/services/query?q={searchTerms}&src={fs:routeTo?}&
&mr={fs:maxResults?}&count={count?}&mt={fs:maxTimeout?}&
p;dn={idn:userDN?}&lat={geo:lat?}&lon={geo:lon?}&radius={
geo:radius?}&bbox={geo:box?}&polygon={geo:polygon?}&dtsta
rt={time:start?}&dtend={time:end?}&dateName={cat:dateName?}&a
mp;filter={fsa:filter?}&sort={fsa:sort?}
```

3. Save the configuration updates

```
config:update
```

4. The new service can now be used in the same way as if it was created using the web administration console.

Directory Permissions

DDF_HOME

DDF_HOME is the directory where DDF is installed.

Windows

Restrict access to sensitive files by ensuring that the only users with access privileges are "Administrators". Right-click the file or directory noted below and follow this path "Full Control" Administrators, "System", and "Creator Owner" for Properties->Security->Advanced) for DDF_HOME (e.g., C:\ddf)

Restrict access to sensitive files by ensuring that only "**System**", and "**Administrators**" have "**Full Control**" to the below files. To do so: right-click on the file or directory below and select Properties->Security->Advanced.

Delete any other groups or users listed with access to DDF_HOME/etc and DDF_HOME/deploy.

*NIX

Protect the DDF from unauthorized access. As root, change the owner and group of critical DDF directories to the NON_ROOT_USER:

A NON_ROOT_USER (e.g., ddf) is recommended for installation.

```
chown -R NON_ROOT_USER $DDF_HOME $DDF_HOME/etc $DDF_HOME/data
chgrp -R NON_ROOT_USER $DDF_HOME/etc $DDF_HOME/data
chmod -R og-w $DDF_HOME/etc $DDF_HOME/data
```

Restrict access to sensitive files by ensuring that the only users with "group" permissions (e.g., ddf-group) have access to the following directories:

Execute the following as the above files (examples assume DDF_HOME is /opt/ddf):

```
chmod -R o /opt/ddf
```

As the the application owner (e.g., `ddf user`), restrict access to sensitive files:

```
chmod 640 /opt/ddf/etc  
chmod 640 /opt/ddf/deploy
```

The system administrator must restrict certain directories to ensure that the application (user) cannot access directories that are not appropriate on the system. For example the `NON_ROOT_USER` should only have read access to `/opt/ddf`.

Deployment Guidelines

DDF relies on the [Directory Permissions](#) of the host platform to protect the integrity of the DDF during operation. System administrators should take the following steps for deploying bundles added to the DDF.

- Prior to allowing an hot deployment, check the available storage space on the system to ensure the deployment will not exceed the available space.
- The maximum storage can be set on the `DDF_HOME/deploy` and `DDF_HOME/system` directories to restrict the amount of space used by deployments.
- Do not assume the deployment is from a trusted source, verify its origination.
- Verify from the source that it is required for deployment on DDF to prevent unnecessary/vulnerable deployments.

Endpoint Schema Validation

SOAP Web Services may have WSDL validation enabled. This step is to ensure that the bundle has WSDL schema validation enabled. These instructions assume the implementation made use of the Spring beans model. All DDF endpoint bundles follow this model.

1. Prior to deploying a bundle/feature verify the schema validation if it is a DDF Endpoint.
2. Modify the `beans.xml` file
 - a. In a terminal (create one if necessary) change directory to the feature directory under the DDF installation directory.

```
cd DDF_HOME/system/com/lmco/ddf/endpoint-bundle.jar
```

- b. Unzip the `endpoint-bundle.jar`

```
unzip endpoint-bundle.jar
```

- c. Change directory to the `beans.xml` file.

```
cd META-INF/spring
```

- d. Open the `beans.xml` file in an editor (e.g., `vi`)
- e. Search for `schema-validation-enabled` and change its value to `true`

```
<entry key="schema-validation-enabled" value="true"/>
```

- f. Save and close the file.
- g. Change directory to the feature directory,

```
cd ../../
```

- 3. Recreate the jar file (use zip or another archive tool)

```
zip endpoint-bundle.jar *
```

- 4. Re-install the feature
 - a. Go into the browser and into the DDF Web Console,
 - b. Click on the install button for the feature.

Schema validation is now enabled for the endpoint.

Security

Managing Users and Passwords

The default security configuration uses a property file located at `DDF_HOME/etc/user.properties` to store users and passwords.

The default Web Administration Console user is "admin" with a password of "admin". Change this password to a more secure password by editing this file.

user.properties

```
Format:
USER=PASSWORD,ROLE1,ROLE2,...

Current default:
admin=admin,admin
```

Enable Password Encryption

In the DDF Text Console enter the following commands:

Enable Password Encryption

```
ddf@local> config:edit --force org.apache.karaf.jaas
ddf@local> config:propset encryption.enabled true
ddf@local> config:update
ddf@local> dev:restart
```

The passwords will then be encrypted in the `users.properties` file once DDF restarts.

Passwords displayed in the admin console

A system administrator must ensure to block visual access to the screen when administering passwords for particular components such as the OpenSearch Source. This is a known issue and will be addressed in a future version of DDF.

DDF Clustering

DDF Application and Configuration Clustering

DDF Application & Configuration Clustering

- DDF Application & Configuration Clustering
 - Introduction
 - Setup of the DDF Cluster
 - Initial Setup
 - Starting and Configuring New DDF Server Node Clusters
 - Adding an Additional DDF Server node to the Cluster
 - Managing Applications
 - Managing Configurations
 - Controlling Feature & Configuration Synchronization
 - Additional Details
 - Configuring DDF Clustering For Unicast & Multicast
 - Verifying Synchronized DDF Nodes
 - Checking For Active Nodes

Introduction

An essential part to the DDF Clustering solution is the ability to manage applications and configurations among cluster nodes. The following documentation will help in setting up a set of DDF server nodes in a cluster, and allow an administrator to manage the applications and configurations that are deployed.

Setup of the DDF Cluster

Initial Setup

The goal of the DDF Cluster solution is to keep applications and configurations synchronized on every DDF server. When setting up a DDF server, it is recommended that only one server is setup per machine (virtual or non-virtual). This means that for each physical machine or virtual machine (VM) that is setup, only one instance of the DDF is deployed on it. This configuration provides the least complexity in configuration of the DDF Cluster and allows the DDF server itself to utilize the resources of the entire system. It is also recommended that all DDF servers and systems be configured the same. This means that all DDF server platforms have the same configurations and applications running initially. Also, ensure that all physical and virtual systems running the DDF servers have the same configuration (e.g. operating system, memory, CPU, etc.).

Starting and Configuring New DDF Server Node Clusters

Before starting your DDF cluster nodes, you must first setup your node network configurations. See the *Configuring DDF Clustering For Multicast & Unicast* section for more information. To view all of the available nodes, navigate in your browser to the DDF Web Console and click on the tab labeled "Clustered Groups". Under the group "default", you should see all running DDF nodes that have been clustered. It is also possible to move all instances into a named group. In order to perform this action, you must have access to one of the DDF command line shells directly or utilize Gogo located at <http://<serverip>8181/system/console/gogo> where *serverip* is the IP address or host name of one of the DDF nodes.

The first action that will be performed is the creation of a new group. To create a new cluster group named "mygroup", execute the following command:

```
cluster:group-create mygroup
```

This command will create a new cluster group which will not contain any nodes. Execute the following command to view all groups:

```
cluster:group-list
```

You should see the following output:

```
Group Members
* [default ] [192.168.1.110:5701* ]
[mygroup ] []
```

As you can see, there are now two groups, the default group and the new group that you have just created. We now need to move the DDF node out of the “default” group and into “mygroup”. Execute the following commands:

```
cluster:group-join mygroup 192.168.1.110:5701

cluster:group-quit default 192.168.1.110:5701
```

The first command allows the DDF node to join the new group. The second command removes the DDF node from the “default” group.

```
Group Members
[default ] []
* [mygroup ] [192.168.1.110:5701* ]
```

These commands should be executed on all “production” nodes located in the default group. Note: The default group will always remain and cannot be deleted. It is possible for DDF nodes to be separated into multiple groups. It is also possible for one DDF node to exist in multiple groups. These are advanced topics and can be addressed in additional documentation links.

Adding an Additional DDF Server node to the Cluster

There may be a need to add an additional DDF server node after an existing DDF cluster has already been configured and deployed. It is important that when adding additional nodes, the new node must match the existing nodes in terms of applications and configurations. Therefore it is good practice to copy one of the existing nodes and push that copy to a new virtual machine or server machine instance. This will provide a stabler transition of the new node into the cluster. Once you have setup the new node, you can follow the instructions above to add the new node into the cluster group, if needed.

Managing Applications

Application management within the cluster has been designed to function as if you were only managing one instance of DDF. All DDF applications are handled at the feature level. Therefore, the Features management console can be used to manage all applications within the DDF Cluster. The Features management console can be accessed by navigating to <http://<serverip>:8181/system/console/features> in a web browser, where *serverip* is the IP address or host name of one of the DDF nodes. You may have to fill in credentials (username and password) to access the console.

From the Features console, the following functions are available:

- Provides a listing of the available features and repositories in the cluster
- Add new repositories to the cluster
- Remove existing repositories from the cluster
- Install features from repositories into the cluster
- Uninstall or remove features from the cluster

For more information on using the console, refer to the DDF User documentation.

Managing Configurations

Just as with Application Management, managing configurations within a cluster was designed to function as if you were configuring one instance of DDF. There are two areas where DDF configurations can be modified. Most modifications will occur within the DDF Configurations

management console. The management console can be accessed by navigating to `http://<serverip>:8181/system/console/configMgr` in a web browser, where `serverip` is the IP address or host name of one of the DDF nodes. You may have to fill in credentials (username and password) to access the console.

From the Configuration console, the following features are available:

- Provides a listing of the available configurations
- Edit configuration values in the cluster
- Unbind the configuration from the bundle
- Remove the configuration from the cluster

For more information on using the console, refer to the DDF User documentation.

Controlling Feature & Configuration Synchronization

There may be instances where certain configurations are to remain local to a certain DDF node. This behavior can be controlled through the cellar groups configuration. To open this configuration, navigate to `http://<serverip>:8181/system/console/configMgr` in a web browser, where `serverip` is the IP address or host name of one of the DDF nodes. You may have to fill in credentials (username and password) to access the console. Within the configuration list, search for the configuration with name “org.apache.karaf.cellar.groups”. Click on the configuration to view / edit. Within the file you will see many configurations listed with the following format:

```
[cluster group name] . [configuration type (e.g. feature, configuration, etc.)]. [list type] = [values]
```

These configurations allow you to control the synchronization of features and configurations through blacklists and whitelists. If you do not want a specific feature or configuration to propagate throughout your cluster group, you can put it into a blacklist. By default for all cluster groups, all features are in the white list:

```
mygroup.features.whitelist.outbound = *
```

with the exception of “cellar”:

```
mygroup.features.blacklist.outbound = cellar
```

As for configurations, by default all configurations are whitelisted with the exception of the following:

```
mygroup.config.blacklist.outbound = org.apache.felix.fileinstall*,  
org.apache.karaf.cellar.groups, org.apache.karaf.cellar.node,  
org.apache.karaf.management, org.apache.karaf.shell, org.ops4j.pax.logging
```

Once you have made your changes, you can save the configuration by pressing the “Save” button.

Additional Details

Configuring DDF Clustering For Unicast & Multicast

By default, DDF clustering utilizes TCP-IP unicast for discovering other DDF nodes. The `hazelcast.xml` file located under `<DDF root>/etc/` contains the port and address configurations for network setup. The TCP-IP unicast mode has been setup to allow for manual configuration and control of initial clustering. This configuration is also beneficial for cases where a particular network cannot support multicast or multicast has been turned off for certain reasons. There is a configuration which allows auto-discovery of DDF nodes and utilizes multicast as a transport. The `hazelcast.xml` file is configured like the following to allow for TCP-IP unicast discovery of cluster nodes:

```

<join>
  <multicast enabled="false">
    <multicast-group>224.2.2.3</multicast-group>
    <multicast-port>54327</multicast-port>
  </multicast>
  <tcp-ip enabled="true">
    <interface>127.0.0.1</interface>
  </tcp-ip>
  <aws enabled="false">
    <access-key>my-access-key</access-key>
    <secret-key>my-secret-key</secret-key>
    <region>us-east-1</region>
  </aws>
</join>

```

As you can see, the multicast option has been set to false and the tcp-ip option is set to true. All systems that will participate in the cluster need to have their ip addresses listed within the interface section highlighted. These modifications must be made for each node. Once these modifications have been made to the hazelcast.xml file, it is recommended that the nodes be restarted.

The following hazelcast.xml configuration would be used for multicast auto-discovery:

```

<join>
  <multicast enabled="true">
    <multicast-group>224.2.2.3</multicast-group>
    <multicast-port>54327</multicast-port>
  </multicast>
  <tcp-ip enabled="false">
    <interface>127.0.0.1</interface>
  </tcp-ip>
  <aws enabled="false">
    <access-key>my-access-key</access-key>
    <secret-key>my-secret-key</secret-key>
    <region>us-east-1</region>
  </aws>
</join>

```

As you can see, the multicast option has been set to true and the tcp-ip option is set to false. A multicast group and port can be specified in the file as highlighted above. These modifications must be made for each node. Once these modifications have been made to the hazelcast.xml file, it is recommended that the nodes be restarted.

Verifying Synchronized DDF Nodes

In most cases, the DDF system console should provide you with a listing of all features, repositories, and configurations that are installed on the cluster. There are times when the cluster can become out of sync. This instance may occur if a system has been offline for some time. One way to verify the synchronized lists of the cluster is to run cluster commands from the command line. In order to perform these actions, you must have access to one of the DDF command line shells directly or through the use of Gogo located at <http://<serverip>8181/system/console/gogo> where *serverip* is the IP address or host name of one of the DDF nodes. Once at the command line, execute the following command to see the list of deployed features for your cluster:

```
cluster:feature-list mygroup
```

This command will list the available features for your cluster group "mygroup".

```
Features for cluster group mygroup
Status Version Name
[installed ] [2.1.0 ] ddf-service-ddms-transformer
[installed ] [2.2.0 ] catalog-opensearch-endpoint
.....
```

To view the cluster group's configurations, execute the following command:

```
cluster:config-list mygroup
```

This command will show all shared configurations among the cluster group "mygroup".

```
-----
Pid: org.ops4j.pax.url.mvn
Properties:
org.ops4j.pax.url.mvn.useFallbackRepositories = false
service.pid = org.ops4j.pax.url.mvn
org.ops4j.pax.url.mvn.disableAether = true
-----

Pid: org.apache.karaf.webconsole
Properties: ....
```

The following command will list all repositories associated with the cluster group "mygroup":

```
cluster:features-url-list mygroup
```

The following will be displayed:

```
mvn:org.apache.cxf.karaf/apache-cxf/2.7.2/xml/features
mvn:org.apache.activemq/activemq-karaf/5.6.0/xml/features
mvn:ddf.catalog.kml/catalog-kml-app/2.1.0/xml/features
mvn:ddf.mime.tika/mime-tika-app/1.0.0/xml/features
```

If for any reason, any of the lists above do not match the list of features, repositories, or configurations found in the DDF system consoles, the following command can be executed:

```
cluster:sync
```

This command should allow for a DDF node to be synchronized with the rest of the cluster.

Checking For Active Nodes

Checking whether a node is active or not can be done utilizing the node ping command. In order to use this command you must have access to one of the the DDF command line shells. A list of nodes can be shown by executing the following command:

```
cluster:node-list
```

The command should show the following output:

```
ID Host Name Port
* [192.168.1.110:5701 ] [192.168.1.110 ] [ 5701]
```

The output will show the ID, host name, and port of each active DDF node in the cluster. The asterisk shows which node you are currently accessing the shell on. Now that you have a listing of node IDs, you can use these to ping other nodes. Execute the following command:

```
cluster:node-ping
```

The following result will print out until you press ctrl-c:

```
PING 192.168.1.110:5701
from 1: req=192.168.1.110:5701 time=9 ms
from 2: req=192.168.1.110:5701 time=4 ms
from 3: req=192.168.1.110:5701 time=2 ms
from 4: req=192.168.1.110:5701 time=3 ms
from 5: req=192.168.1.110:5701 time=4 ms
from 6: req=192.168.1.110:5701 time=2 ms
from 7: req=192.168.1.110:5701 time=2 ms
^C
```

The output will provide you with a typical ping result showing connectivity and response times.

DDF Load Balancer

DDF Load Balancer

Contained within DDF is a Load Balancer utility that allows incoming traffic to be distributed over multiple instances of DDF. The DDF Load Balancer supports two protocols, HTTP and HTTPS. The Load Balancer can be configured to run either protocol or both at the same time. The DDF Load Balancer has been configured to utilize a "Round Robin" algorithm to distribute transactions. The load balancer is also equipped with a failover mechanism. When the load balancer attempts to access a server that is non-functional, it will receive an exception and move on to the next server on the list to complete the transaction.

Setting up the DDF Load Balancer

There are two methods for standing up a DDF Load Balancer. The standalone load balancer distribution that is pre-built may be used or the load balancer can be added to the full DDF distribution.

Setting up the Pre-Built Standalone DDF Load Balancer

When a DDF build occurs, a distribution of the load balancer is created on top of a lightweight version of DDF. After building DDF, this distribution can be found under <DDF-BUILD>/distribution/ddf-standalone-loadbalancer/target. The build file will be an archive called standalone-loadbalancer.zip. Unzipping this file into a directory will provide you with a DDF instance specifically designed to only run the load balancer. This instance will have the DDF Load Balancer already installed and running. Start the DDF instance as normal.

Setting up the DDF Load Balancer On the Full Distribution

The following steps will help in getting the load balancer to run on a full DDF distribution. It is recommended that the load balancer run on a dedicated machine and not be utilized for any other DDF service activity other than for load balancing.

- Start your instance of the DDF framework
- Open the DDF management console by navigating to `http://<serverip>:8181/system/console/features` in a web browser, where *serverip* is the IP address or host name of your DDF instance. You may have to fill in credentials (username and password) to access the console.
- Within the "Features" table on the current page, scroll down until you see a feature labeled "camel-jetty". Install this feature by clicking on the circular button located at the far right of the current row.
- Navigate back to the top of the page and click on the "Bundles" tab in the main navigation.
- To install the load balancer bundle, click the "Install/Update" button located just above the table.
- Click the checkbox for start bundle and then click on the "choose file" button. From here, select the jar file located under `<DDF_INSTALL_DIRECTORY>/cloud/loadbalancer/target/cloud-loadbalancer.jar`.
- Click the "Open" button to select it and then click the "Install or Upgrade" button.
- You should now see the service marked active in the table.

Configuring the DDF Load Balancer

The DDF Load Balancer can be configured to allow for multiple DDF nodes to be balanced across it. It can also be configured with a particular port on which to accept connections. Configurations differ slightly between the HTTP and HTTPS based load balancers. All configurations are dynamic in that configuration settings are immediately applied and no restart of DDF is necessary.

Configuring the HTTP Load Balancer

To access the load balancer configuration, follow the steps below:

- From the DDF management console, click on the "Configuration" tab in the main navigation.
- Within the configuration table, scroll down to the configuration entry labeled *Platform HTTP Load Balancer*. There will be two entries. Click on the 1st entry that is in non-italics.
- The configuration for the load balancer contains two fields: **Load Balancer Port** and **IP Address and Port**.
- For Load Balancer Port, enter the port number which you would like to access on your load balancer to reach the other systems.
- For IP Address and Port, enter a comma delimited list of IP address and port for each DDF node that will be balanced. The format for IP address and port is the following: `<IP_ADDRESS>:<PORT>` (e.g. 192.168.1.123:8181,192.168.1.22:8181)
- Once all configurations have been added, click the save button.

At this point, the load balancer should reset and should be ready to accept requests. These configurations can be updated at any time without starting the host DDF instance.

Configuring the HTTPS Load Balancer

It is possible to run the HTTPS load balancer by itself or run it in parallel with the HTTP load balancer. The HTTPS load balancer utilizes the centralized SSL configurations within DDF, along with the load balancer configurations. To configure the HTTPS load balancer, follow the steps below:

- First step is to find the SSL configurations and check that the values are correct.
- From the DDF management console, click on the "Configuration" tab in the main navigation.
- Within the configuration table, scroll down to the configuration entry labeled *Pax Web Runtime*. Click on this entry.
- Ensure that these settings are in sync with what you have configured for your DDF nodes that will be balanced. Save settings or close window.
- Next, within the configuration table, scroll down to the configuration entry labeled *Platform HTTPS Load Balancer*. There will be two entries. Click on the 1st entry that is in non-italics.
- The configuration for the load balancer contains three fields: **Load Balancer Host**, **Load Balancer Port** and **IP Address and Port**.
- For Load Balancer Host, enter the host name or IP address that should be used for the host load balancer machine.
- For Load Balancer Port, enter the port number which you would like to access on your load balancer to reach the other systems.
- For IP Address and Port, enter a comma delimited list of IP address and port for each DDF node that will be balanced. The format for IP address and port is the following: `<IP_ADDRESS>:<PORT>` (e.g. 192.168.1.123:8993,192.168.1.22:8993)

- Once all configurations have been added, click the save button.

Note: Since SSL requests will be coming from a client into the load balancer, it is essential that the nodes being balanced have the same security policy and settings. The client has no idea which DDF server it will be connecting with behind the load balancer. The client is responsible for connecting securely with the load balancer and the load balancer is responsible for connecting securely and consistently with all DDF nodes.

Note: The DDF Load Balancer cannot run on the same port as the DDF Web Console or other web services. If you would like the load balancer to run on this port, you must change the web console port to a different port number. This configuration parameter can be found in the "Pax Web Runtime" configuration.

DDF Data Migration

Description

Data Migration is the process of moving metadata from one catalog provider to another. It is also the process of translating metadata from one format to another. Data migration is necessary when a user decides to use metadata from one catalog provider in another catalog provider. The instructions will show how to transfer metadata from one catalog provider to another catalog provider. In addition, the instructions will show how to convert metadata to different data formats.

Setup

Start DDF as instructed at [Starting and Stopping](#)

Move Metadata from Catalog Provider to Another Catalog Provider

Export metadata out of Catalog Provider

1. Configure a desired Catalog Provider
2. From the command line of DDF console, use the dump command to export all metadata from catalog provider into serialized data files. The block below shows a command for running on Linux and a command for running on Windows.

```
ddf@local
dump "/myDirectory/exportFolder"
or
dump "C:/myDirectory/exportFolder"
```

Ingest exported metadata into Another Catalog Provider

1. Configure a different Catalog Provider
2. From the command line of DDF console, use the ingest command to import exported metadata from serialized data files into catalog provider. The block below shows a command for running on Linux and a command for running on Windows.

```
ddf@local
ingest -p "/myDirectory/exportFolder"
or
ingest -p "C:/myDirectory/exportFolder"
```

Translate Metadata from one Format to Another

Metadata can be converted from one data format to another format. Only the data format would change but the content of the metadata do not as long as the option -p is use with the ingest command. The process for converting metadata is by ingest a data file into a catalog provider in one format and dump it out into a file in another format. Additional information for ingest and dump commands are at [Catalog Commands](#).

Users Guide Appendix

Contents:

- [Software Versioning](#)
- [FAQ, DDF Use of OGC Filter](#) — answers frequently asked questions about the use of the OGC Filter in DDF Catalog Queries and Subscriptions
- [Integrator Scenarios](#)
- [Common Problems and Solutions](#)
- [DDF Directory Contents after Installation](#)

Software Versioning

Format

DDF component versions take the form *major.minor[.maintenance[.build]]*

Major

A major release occurs when new functionality is added to DDF that requires external clients to modify their implementations that interface with DDF.

Minor

A minor release occurs when new functionality is added to DDF that may require DDF developers to modify their implementations, but does not require external clients to modify their implementations that interface with DDF.

Maintenance

A maintenance release occurs when bugs are fixed in DDF but do not result in external clients or developers having to change their implementations that interface to DDF.

Build

The build section indicates milestone or maturity (e.g. ALPHA, M1, RC, SNAPSHOT)

M = Milestone

RC = Release Candidate

Thirdparty Versioning

Sometimes it is necessary to modify jars in order to make them bundles that will run in the OSGi container. These thirdparty jars should have the version of the JAR that it is wrapping followed by a suffix of "_X" where X is the version of that pom. For instance, if the artifact is wrapping the JTS jar of version 1.11, the version of that pom creating the bundle would be 1.11_1. If that specific pom is ever changed, the suffix should be incremented. For example, a subsequent change would make it 1.11_2 and then 1.11_3 on the next change.

FAQ, DDF Use of OGC Filter

Frequently Asked Questions

- [What is an OGC Filter?](#)
- [Why does DDF need an OGC Filter?](#)
- [If the standard is implemented using XML, how will it be used in DDF Catalog which is mostly implemented in Java?](#)

What is an OGC Filter?

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

Why does DDF need an OGC Filter?

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 can 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 being able to Add or Divide expressions in a query, but that is not the intended use for DDF.

If the standard is implemented using XML, how will it be used in DDF Catalog which is mostly implemented in Java?

The DDF Catalog Framework utilizes the implementation provided by Geotools that is 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. A special note should be added that there is no intent right now that Java developers should be parsing or viewing the XML representation. The intention is that developers will use the Java objects exclusively to complete query tasks.

Integrator Scenarios

Overview

This section describes several scenarios that an Integrator may commonly encounter in configuring DDF.

Note that the configuration steps can be done in the [Web Console](#) or the [System Console](#).

- [Configuring DDF as a Fanout Proxy](#)
- [Reconfiguring DDF with a Different Catalog Provider](#)

Configuring DDF as a Fanout Proxy

Description

This scenario describes how to configure DDF as a fanout proxy such that only queries and resource retrieval requests are processed and create/update/delete requests are rejected. Additionally, all queries are enterprise queries and no catalog provider needs to be configured.

Steps

1. Start DDF following the [Starting and Stopping](#) instructions.
2. Reconfigure DDF in fanout proxy mode by going to the Features tab in the Admin Console. The [Standard Catalog Framework](#) (catalog-core-standardframework feature) should be uninstalled first, then install the [Catalog Fanout Framework App](#) (catalog-core-fanoutframework feature), following the instructions in the [Installing Features](#) section of the Administrator's Guide.

DDF is now operating as a fanout proxy. Only queries and resource retrieval requests will be allowed. All queries will be federated. Create, update, and delete requests will throw an `UnsupportedOperationException`, even if a [Catalog Provider](#) was configured previous to the reconfiguration to fanout.

3. Verify DDF is in fanout configuration by going to the Services tab in the Admin Console. In the far right column labeled Bundle, there should be entries near the top called `fanout-catalog-framework` for the [Catalog Framework](#), [Event Processor](#), and [Catalog Plugins](#), similar to what is depicted below. (Note that service IDs and Bundle IDs will vary.)

Services in Admin Console for fanout

Id	Type(s)	Bundle
320	[ddf.catalog.CatalogFramework]	
fanout-catalogframework (174)		
318	[ddf.catalog.event.EventProcessor]	
fanout-catalogframework (174)		
316	[ddf.catalog.federation.FederationStrategy]	
fanout-catalogframework (174)		
319	[ddf.catalog.federation.FederationStrategy]	
fanout-catalogframework (174)		
317	[ddf.catalog.plugin.PostIngestPlugin]	
fanout-catalogframework (174)		

Reconfiguring DDF with a Different Catalog Provider

Description

This scenario describes how to reconfigure DDF to use a different catalog provider.

This scenario assumes DDF is already running.

Use of the Dummy Catalog Provider

This scenario uses the Dummy Catalog Provider as the catalog provider DDF is being reconfigured to use. This is because the Dummy Catalog Provider is the only other catalog provider shipped with DDF out of the box. The Dummy Catalog Provider should **never** be used in a production environment. It is only used for testing purposes.

Steps

1. Uninstall a Catalog Provider (if installed) following the instructions in the [Uninstalling Features](#) section.
2. Install the new [Catalog Provider](#), which will be the [Dummy Provider](#), by installing its feature `ddf-provider-dummy`, following the instructions in the [Installing Features](#) section.
3. Verify DDF is running with the [Dummy Provider](#) as its [Catalog Provider](#) by going to the Services tab in the Admin Console. In the far right column labeled Bundle, there should be an entry labeled `ddf.providers.provider-dummy`, as shown below.

New catalog provider Dummy Provider installed

Id	Type(s)	Bundle
325	[ddf.catalog.source.CatalogProvider]	
ddf.providers.provider-dummy (175)		
	osgi.service.blueprint.compname DummyProvider	

Common Problems and Solutions

- [Blank Web Console](#)
- [CXF BusException](#)
- [Distribution does not start](#)
- [Exception Starting DDF](#)
- [DDF Becomes Unresponsive to Incoming Requests](#)

Blank Web Console

Blank Web Console

Problem: <http://localhost:8181/system/console> comes up as a blank page.

Solution: Restart DDF

- 1. Shutdown DDF from its console

```
ddf@local>shutdown
```

- 2. Start DDF back up with the script

```
./ddf
```

- Verify that all of the files were copied over correctly during the *Deploy Bundles* step.

CXF BusException

CXF BusException

Problem: The following Exception is thrown:

```
org.apache.cxf.BusException: No conduit initiator
```

Solution: Restart DDF

- Shutdown

```
ddf@local>shutdown
```

- Startup

```
./ddf
```

Distribution does not start

Problem

DDF will not start up when calling the start script defined during installation.

Solution

1. Verify that Java is correctly installed.

```
java -version
```

Should return something similar to

```
java version "1.6.0_22" Java(TM) SE Runtime Environment (build
1.6.0_22-b04) Java HotSpot(TM) Server VM (build 17.1-b03, mixed
mode)
```

2. If running *nix verify that bash is installed.

```
echo $SHELL
```

Should return

```
/bin/bash
```

Exception Starting DDF

Exception Starting DDF

Problem: Following exception is thrown starting DDF on a Windows machine (x86)

Using an unsupported terminal: java.lang.NoClassDefFoundError: Could not initialize class org.fusesource.jansi.internal.Kernel32

Solution: *Install missing Windows libraries.*

Some Windows platforms are missing libraries needed by DDF. These libraries are provided by the [Microsoft Visual C++ 2008 Redistributable Package x64](#).

DDF Becomes Unresponsive to Incoming Requests

Symptoms

- Multiple java.exe processes running indicating more than one DDF instance is running. This can be caused by not shutting down another DDF.

DDF becomes Unresponsive to Incoming Requests

Problem: DDF Becomes Unresponsive to Incoming Requests. An example of what the log file will look like is located below:

```
Feb 7, 2013 10:51:33 AM org.apache.karaf.main.SimpleFileLock lock
INFO: locking
Feb 7, 2013 10:51:33 AM org.apache.karaf.main.Main doLock
INFO: Waiting for the lock ...
Feb 7, 2013 10:51:33 AM org.apache.karaf.main.SimpleFileLock lock
INFO: locking
Feb 7, 2013 10:51:33 AM org.apache.karaf.main.Main doLock
INFO: Waiting for the lock ...
Feb 7, 2013 10:51:34 AM org.apache.karaf.main.SimpleFileLock lock
INFO: locking
Feb 7, 2013 10:51:34 AM org.apache.karaf.main.SimpleFileLock lock
INFO: locking
Feb 7, 2013 10:51:35 AM org.apache.karaf.main.SimpleFileLock lock
INFO: locking
Feb 7, 2013 10:51:35 AM org.apache.karaf.main.SimpleFileLock lock
INFO: locking
```

Possible Solutions:

- Wait for proper shutdown of DDF prior to starting a new instance

- Verify running java.exe are not DDF (e.g. kill/close if necessary)
- Utilize automated [start/stop scripts](#) to run DDF as a service.

DDF Directory Contents after Installation

During DDF installation, the following major directories (displayed in the table below) will be created, modified, or replaced in the destination directory.

Directory Name	Description
bin	Scripts to start and stop DDF
data	The working directory of the system – installed bundles and their data
data/log/ddf.log	Log file for DDF, logging all errors, warnings, and (optionally) debug statements. This log rolls up to 10 times, frequency based on a configurable setting (default=1 MB)
deploy	Hot-deploy directory – KARs and bundles added to this directory will be hot-deployed (Empty upon DDF installation)
docs	The DDF Catalog API Javadoc
etc	Directory monitored for addition/modification/deletion of third party .cfg configuration files
etc/ddf	Directory monitored for addition/modification/deletion of DDF-related .cfg configuration files (e.g., Schematron configuration file)
etc/templates	Template .cfg files for use in configuring DDF sources, settings, etc., by copying to the etc/ddf directory.
lib	The system's bootstrap libraries. Includes the ddf-branding.jar file which is used to brand the system console with the DDF logo.
licenses	Licensing information related to the system
system	Local bundle repository. Contains all of the JARs required by DDF, including third-party JARs.

DDF Developer's Guide

Table of Contents

- [Developer's Documentation Guide](#) — Establishes conventions used throughout the documentation
- [Ensuring Compatibility](#)
- [Development Recommendations](#) — provides general developer tips and recommendations for OSGi bundle development
- [Working with OSGi](#) — quick tutorials of OSGi basic concepts
- [Developing Catalog Components](#) — Describes how to create Catalog components. Used in conjunction with the Javadoc to begin extending the DDF Catalog.
- [Developing at the Framework Level](#) — Help with framework concepts and development.
- [Developing Action Components](#) — Describes how and why to create Action Components.
- [Developing for the DDF Marketplace](#)
- [Developing Security Token Service Components](#)
- [Building](#)
- [DDF Camel Components](#)

Introduction

This guide discusses the several extension points and components permitted by the Distributed Data Framework (DDF) Catalog API. Using code examples, diagrams, and references to specific instances of the Catalog API, this guide will provide details on how to develop and integrate various DDF components.

Development Prerequisites

It is recommended to understand the [DDF Catalog](#) fully before beginning development.

DDF is almost completely 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:

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

Documentation on OSGi can be viewed at the OSGi Alliance website (<http://www.osgi.org>). Books that some OSGi beginners found helpful are OSGi and Apache Felix 3.0 Beginner's Guide and OSGi in Action: Creating Modular Applications in Java. For specific code examples from DDF, source code can be seen in the [Working with OSGi](#) section.

Getting Setup

All that is necessary to develop on DDF is either access to the source code ([Building DDF Offline from Source](#)) or the DDF binary zip file `ddf-2.0.0.zip`.

Integrated Development Environment (IDE)

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

Directory Structure

During DDF installation, the following major directories (displayed in the table below) will be created, modified, or replaced in the destination directory.

Directory Name	Description
<code>bin</code>	Scripts to start and stop DDF
<code>data</code>	The working directory of the system – installed bundles and their data
<code>data/log/ddf.log</code>	Log file for DDF, logging all errors, warnings, and (optionally) debug statements. This log rolls up to 10 times, frequency based on a configurable setting (default=1 MB)
<code>deploy</code>	Hot-deploy directory – KARs and bundles added to this directory will be hot-deployed (Empty upon DDF installation)
<code>docs</code>	The DDF Catalog API Javadoc
<code>etc</code>	Directory monitored for addition/modification/deletion of third party <code>.cfg</code> configuration files
<code>etc/ddf</code>	Directory monitored for addition/modification/deletion of DDF-related <code>.cfg</code> configuration files (e.g., Schematron configuration file)
<code>etc/templates</code>	Template <code>.cfg</code> files for use in configuring DDF sources, settings, etc., by copying to the <code>etc/ddf</code> directory.
<code>lib</code>	The system's bootstrap libraries. Includes the <code>ddf-branding.jar</code> file which is used to brand the system console with the DDF logo.
<code>licenses</code>	Licensing information related to the system
<code>system</code>	Local bundle repository. Contains all of the JARs required by DDF, including third-party JARs.

Additional Documentation

Additional documentation on developing with the core technologies used by DDF can be found on their respective websites. Notably:

1. Karaf <http://karaf.apache.org/manual/latest-2.2.x/developers-guide/index.html>
2. CXF <http://cxf.apache.org/docs/overview.html>
3. Geotools <http://docs.geotools.org/latest/developer/>

Developer's Documentation Guide

Documentation Updates

The most current Distributed Data Framework (DDF) documentation is available at <https://tools.codice.org/wiki/display/DDF/>.

Questions

Questions about DDF or this documentation should be posted to the ddf-users forum (<https://groups.google.com/d/forum/ddf-users>), ddf-announcements forum (<https://groups.google.com/d/forum/ddf-announcements>), or ddf-developers forum (<https://groups.google.com/d/forum/ddf-developers>) where they will be responded to quickly by a member of the DDF team.

Conventions

The following conventions are used within this documentation:

This is a **Tip**, used to provide helpful information.

This is an **Informational Note**, used to emphasize points, remind users of beneficial information, or indicate minor problems in the outcome of an operation.

This is an **Emphasized Note**, used to inform of important information.

This is a **Warning**, used to alert users about the possibility of an undesirable outcome or condition.

Customizable Values

Many values used in descriptions are customizable and should be changed for specific use cases. These values are denoted by `< >`, and by `[[]]` when within XML syntax. When using a real value, the placeholder characters should be omitted.

Code Values

Java objects, lines of code, or file properties are denoted with the Monospace font style. Example: `ddf.catalog.CatalogFramework`

Hyperlinks

Some hyperlinks (e.g., <http://localhost:8181/system/console>) within the documentation assume a locally running installation of DDF. Simply change the hostname if accessing a remote host.

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 be aware when and where they establish 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 and Web Console (for Metatype support)

Avoid developing dependencies on the implementations directly, as compatibility in future releases cannot be guaranteed.

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.

DDF Software Versioning

DDF follows the [Semantic Versioning](#) White Paper for bundle versioning (see [Software Versioning](#)).

Third Party and Utility Bundles

It is recommended to avoid building directly on included third party and utility bundles. These components do provide utility (e.g. JScience) 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 upgrade CXF frequently between releases to take advantage of new features. If building on these components, please 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 like commons-geospatial, 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).

Best Practices

- Always use a version number when exporting a package

Export Example

```
<Export-Package>  
  ${project.artifactId}*;version=${project.version}  
</Export-Package>
```

In the example above the `${project.artifactId}` represents the project and artifactId of the package being exported. The `${project.version}` represents the version of the project.

- Try to avoid deploying multiple versions of a bundle. Although OSGi is designed to support multiple versions other developers may leave the version off packages that are being imported. If the bundle is versioned and designed appropriately typically multiple versions of the bundle will not be an issue. However, if each bundle is competing for a specific resource then race conditions may occur. Third party and utility bundles (often denoted by commons in the bundle name) are the general exception to this rule, as these bundles will likely function as expected with multiple versions deployed.

Development Recommendations

Overview

This section provides general developer tips and recommendations for OSGi bundle development.

Dependency Injection Frameworks

It is highly recommended to use a dependency injection framework such as Blueprint, Spring-DM, or iPojo for non-advanced OSGi tasks. Dependency injection frameworks allow for more modularity in code, keep the code's business logic clean of OSGi implementation details, and take the complexity out of the dynamic nature of OSGi. In OSGi, services can be added and removed at any time, and dependency injection frameworks are better suited to handle these types of situations. Allowing the code to be clean of OSGi packages also makes code easier to reuse outside of OSGi. These frameworks provide code conveniences of service registration, service tracking, configuration property management, and other OSGi core principles.

Basic Security

Provided by [Pierre Parrend](http://www.slideshare.net/kaihackbarth/security-in-osgi-applications-robust-osgi-platforms-secure-bundles) (<http://www.slideshare.net/kaihackbarth/security-in-osgi-applications-robust-osgi-platforms-secure-bundles>)

- Bundles should
 - Never use **synchronized** statements that rely on third party code. Keep in mind multi-threaded code when using synchronized statements in general as they can lead to performance issues.
 - Only have dependencies on bundles that are trusted.
- Shared Code
 - Provide only final static non-mutable fields.
 - Set security manager calls during creation in all required places at the beginning of methods.
 - All Constructors
 - *clone()* method if a class implements *Cloneable*
 - *readObject(ObjectInputStream)* if the class implements *Serializable*
 - Have security check in **final** methods only.
- Shared Objects (OSGi services)
 - Only have basic types and **serializable final** types as parameters.
 - Perform copy and validation (e.g. null checks) of parameters prior to using them.
 - Do not use **Exception** objects that carry any configuration information.

Working with OSGi

OSGi Usage in DDF

- [OSGi Service Registry](#)
 - [Spring DM - Retrieving a Service instance](#)
 - [Blueprint - Retrieving a Service instance](#)
 - [Blueprint - Registering a Service into the Registry](#)
 - [Packaging Capabilities as Bundles](#)
 - [Creating a Bundle](#)
 - [Maven Bundle Plugin](#)
 - [Deploying a Bundle](#)
 - [Verifying Bundle State](#)
 - [Additional Resources](#)
-

OSGi Service Registry

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. There are many tutorials and guides available on the Internet for both of these frameworks. Refer to the [Additional Resources](#) section for details not covered in this guide. Links to some of these guides are given in the External Links area of this section.

Spring DM - Retrieving a Service instance

Spring DM example of retrieving and injecting Services

```
1 <beans xmlns="http://www.springframework.org/schema/beans"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns:osgi="http://www.springframework.org/schema/osgi">
4
5   <osgi:reference id="ddfCatalogFramework"
interface="ddf.catalog.CatalogFramework" />
6
7   <bean class="my.sample.NiftyEndpoint">
8     <constructor-arg ref="ddfCatalogFramework" />
9   </bean>
10 </beans>
```

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

Blueprint - Retrieving a Service instance

Blueprint example of retrieving and injecting Services

```
1 <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
2
3   <reference id="ddfCatalogFramework"
interface="ddf.catalog.CatalogFramework" />
4
5   <bean class="my.sample.NiftyEndpoint" >
6     <argument ref="ddfCatalogFramework" />
7   </bean>
8
9 </blueprint>
```

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

Blueprint - Registering a Service into the Registry

Creating a bean and registering it into the Service Registry

```
1 <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
2
3   <bean id="transformer" class="my.sample.NiftyTransformer"/>
4
5   <service ref="transformer"
6     interface="ddf.catalog.transform.QueryResponseTransformer" />
7 </blueprint>
```

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

Packaging Capabilities as Bundles

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

Alternative bundle creation methods

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

See external links (below) for resources that give in-depth guides on creating bundles.

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.

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>${project.name}</Bundle-Name>
        <Export-Package />

        <Bundle-SymbolicName>${project.groupId}.${project.artifactId}</Bundle-SymbolicName>
        <Import-Package>
          ddf.catalog,
          ddf.catalog.*
        </Import-Package>
      </instructions>
    </configuration>
  </plugin>
...
</build>
...
```

Deploying a Bundle

A bundle is typically installed in two ways:

- As a feature.
- Hot deployed in the /deploy directory.

The fastest way during development to deploy a created bundle 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 the CAB Client is deployed and running.

Verifying with grep

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

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

Additional 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>

Developing Catalog Components

- [Catalog Development Fundamentals](#) — Introduces the fundamentals of working with the Catalog API, including Metacards and use of the OGC Filter for Queries
- [Developing an Endpoint](#) — Describes how to create an Endpoint and the description of methods an Endpoint can invoke.
- [Developing a Catalog Plugin](#) — Describes the different plugins that can be created.
- [Developing a Source](#) — Describes the different types of sources and how to create them.
- [Developing Transformers](#) — Describes the different types of transformers and how to create them.
- [Developing a Resource Reader](#) — Describes how to create a ResourceReader.
- [Developing a Resource Writer](#)
- [Developing a Registry Client](#) — Briefly describes how Registry Clients create Sources within the framework.

Catalog Development Fundamentals

- [Simple Catalog API Implementations](#) — Describes the simple implementations that are included out-of-the-box with DDF.
- [Use of the Whiteboard Design Pattern](#) — Introduces the Whiteboard Design Pattern and describes its use in OSGi.
- [Working with the Catalog Framework](#) — Provides an overview and usage of the Catalog Framework
- [Working with Metacards](#) — Overview of how Metacards are used, created, and processed
- [Working with Queries](#) — Overview of Query objects such as what they are, how they are created, and how are they processed.
- [Working with Subscriptions](#) — Description of how to create and use subscriptions with ApplicationName Eventing.
- [Working with Filters](#)
- [Working with Transformers](#) — Description and usage of what transformers are and how they are used.
- [Working with Resources](#) — Description of retrieval and storage of resources.
- [Commons-DDF Utilities](#) — Explains this bundle's classes and usage.
- [Working with Settings](#)

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, then the developer can at least have a working code reference to base future development.

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.

Working with the Catalog Framework

- [Overview](#)
- [Usage](#)
 - [Catalog Framework Reference](#)
- [Methods](#)
 - [Create, Update, and Delete](#)
 - [Query](#)
 - [Resources](#)
 - [Sources](#)
 - [Transforms](#)

Overview

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

Usage

Catalog Framework Reference

The Catalog Framework can be requested from the OSGi registry. See [Working with OSGi](#) for more details on Blueprint injection.

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 `PreIngestPlugins` before execution and `PostIngestPlugins` 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 `PreQueryPlugins` before execution and `PostQueryPlugins` after execution.

Resources

[Resource](#) methods retrieve products from Sources.

Resource Methods

```
public ResourceResponse getEnterpriseResource(ResourceRequest request)
throws IOException, 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 `PreResourcePlugins` before execution and `PostResourcePlugins` 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;
```

Working with Metacards

Working with Metacards

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. For more information, see [Data Components](#) in the Users Guide.

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 `Attributes` without having to create a new `MetacardType`. `MetacardImpl` uses `BASIC_METACARD` as its `MetacardType` (see below).

Metacard Extensibility

Often times the `BASIC_METACARD` `MetacardType` does not provide all the functionality or attributes necessary for a specific task. Sometimes for performance or convenience purposes it is necessary to create custom attributes even if others will not be aware of those attributes. One example would 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 `AttributeDescriptor` for each `Attribute`, indicating its key and value type. `AttributeType` support is limited to those types defined by the Catalog, as documented in [Data Components](#) in the Users Guide.

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

Metacard Type Registry

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.

`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 lookup 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.

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` in 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.

BasicTypes

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.
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.

Working with Queries

Overview

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. See the [Creating Filters](#) section for more on Filters.

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.

Creation

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, and 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);

```

Evaluation

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.

Additional Reading

- [Creating Filters](#)
- [Parsing Filters](#)

Working with Subscriptions

Overview

The Eventing capability of DDF allows endpoints (and thus external users) to create a "standing query" and be notified when a matching Metacard is created, updated, or deleted.

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 hostility 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 keep an eye out for 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 only return records containing the keyword "pirate." How these event notifications are handled and processed is up to the client's event consumer which receives all callbacks from DDF. With the subscription in place, the analysts will be notified only of metadata related to the pirating activity, so analysts can obtain better situational awareness.

The key components of DDF Eventing include:

- [Subscription](#) - represent "standing queries" in the Catalog.
- [Delivery Method](#) - provides the operation (created, updated, deleted) for how an event's metacard can be delivered.
- [Event Processor](#) - provides an engine that creates, updates, and deletes subscriptions for event notification.
- Event Consumer - service that receives and processes event notifications on the client side.
- Callback URL - location of Event Consumer.

These components can be studied in more depth by referring to their corresponding sections in the Integrator's Guide.

This section discusses how a developer can create subscriptions and event consumers to work with DDF Eventing. More information about DDF Eventing in general can be found in the [Eventing](#) section of the Integrator's Guide.

Creating a Subscription

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. See the [Creating Filters](#) section of the Developer's Guide for more information on OGC Filters.

The functionality of these overridden methods is typically delegated to whatever `Filter` implementation the `Subscription` is using.

The developer will also need to define `getDeliveryMethod`. This method should return an instance of the appropriate `DeliveryMethod` that will communicate to the event consumer designated to receive the notifications.

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`:

SubscriptionImpl

```
public class SubscriptionImpl implements Subscription{

    private Filter filter; //Subscription generally contains a Filter to
    perform the Filter methods against
    private DeliveryMethod deliveryMethod; //the DeliveryMethod instance to
    handle events and call out to event consumer

    public SubscriptionImpl(Filter filter, DeliveryMethod deliveryMethod){
        this.filter = filter;
        this.deliveryMethod = deliveryMethod;
    }

    @Override
    public boolean evaluate(Object object) {
        //evaluates incoming object against filter criteria
    }

    @Override
    public Object accept(FilterVisitor visitor, Object extraData) {
        //visits filter in order to read and parse it
    }

    @Override
    public Set<String> getSourceIds() {
        //returns source ids that subscription should watch for events
    }

    @Override
    public boolean isEnterprise() {
        //should this subscription watch for events from the entire
        enterprise?
    }

    @Override
    public DeliveryMethod getDeliveryMethod() {
        return deliveryMethod;
    }
}
```

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 kept track 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> subscriptions = new HashMap<String,
ServiceRegistration>();

//Instance of DeliveryMethod with the callback URL of the event consumer
DeliveryMethod deliveryMethod = new DeliveryMethodImpl(String callbackURL);

//Creates a filter that matches on the Metacard ID
FilterFactory filterFactory = new FilterFactoryImpl();
Filter filter = filterFactory.equals( filterFactory.property( Metacard.ID
), filterFactory.literal( id ) );

//New Subscription instance
Subscription subscription = new SubscriptionImpl(filter, deliveryMethod);

//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 serviceRegistration = context.registerService(
"ddf.catalog.event.Subscription", subscription , properties );
subscriptions.put(subscriptionId, serviceRegistration);
```

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 `DeliveryMethod` implementation typically contains a callback URL. It uses this URL to know where the event consumer is located.

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 a little 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 `DeliveryMethod` knows how to communicate with the event consumer. It also knows what it is expecting. Each of the aforementioned methods should call out to an event consumer. A `DeliveryMethod` reports to the event consumer the metacards involved in the event, what type of event occurred, and possibly the subscription ID if of interest in whatever format the event consumer requires. The following is an implementation stub for `DeliveryMethod`:

DeliveryMethodImpl

```
public class DeliveryMethodImpl implements DeliveryMethod {

    @Override
    public void created(Metacard newMetacard) {
        //Sends created notification to event consumer
    }

    @Override
    public void updatedHit(Metacard newMetacard, Metacard oldMetacard) {
        //Sends updatedHit notification to event consumer
    }

    @Override
    public void updatedMiss(Metacard newMetacard, Metacard oldMetacard) {
        //Sends updatedMiss notification to event consumer
    }

    @Override
    public void deleted(Metacard oldMetacard) {
        //Sends deleted notification to event consumer
    }
}
```

Creating an Event Consumer

An event consumer is a service designed to receive and process event notifications. This event consumer can be any type of service running at a location accessible to DDF. If the service is a SOAP service, the `DeliveryMethod` will need to communicate with it according to its WSDL. On the other hand, if it is a REST service, the `DeliveryMethod` will invoke HTTP operations on the callback URL. Typically, whatever type of service the event consumer is, it will have operations corresponding to each of the created, updated, and deleted events. This way, the consumer will know what type of event occurred on DDF.

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 sr = (ServiceRegistration)
subscriptions.get(subscriptionId);

//Unregister Subscription from OSGi Service Registry
sr.unregister();

//Remove Subscription from Map keeping track of registered Subscriptions.
subscriptions.remove(subscriptionId);
```

Working with Filters

- [Creating Filters](#)
- [Parsing Filters](#)
- [Developer Use of OGC Filter](#) — Answers frequently asked questions about the use of the OGC Filter in DDF Catalog Queries and Subscriptions
- [DDF Filter Profile](#)

Creating Filters

Overview

An OGC Filter is a [Open Geospatial Consortium \(OGC\) standard](#) 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](#). 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. Developers instead solely use 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
```

Usage

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`.

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

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(String
caseSensitiveContextualSearchPhrase);
FilterBuilder.attribute(attributeName).is().like().fuzzyText(String
fuzzySearchPhrase);
```

Directly implementing the Filter (advanced)

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 [04-095](#), units of measure can be expressed as a URI. In order 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. The `UomOgcMapping` essentially maps the OGC [Symbology Encoding](#) standard URI's 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.

Creation

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`.

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 `PropertyIsLike` OGC 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 characters 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.



Unknown macro: 'plantuml'

XML View of Example Creating-Filters-3

Another way to view this type of Filter is through an XML model as 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 flag to Sources when to do a fuzzy search. The syntax expected by providers is similar to the Fuzzy Function. Usage example below.

Fuzzy Function Usage

```
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);
```

Additional Reading

- [Oracle Catalog Provider Implementation Details](#)
- [Developer Use of OGC Filter](#)

Parsing Filters

Overview

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" (04-095).

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 [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` might 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 Parsing-Filters1



Unknown macro: 'plantuml'

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, then the `FilterDelegate` could yield an expression such as

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



Unknown macro: 'plantuml'

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;
    ...
}
```

Developer Use of OGC Filter

Related Topic: [FAQ, DDF Use of OGC Filter in the Integrator's Guide](#)

Frequently Asked Questions

- [How can I create a filter?](#)
- [I see there is a Geotools Query, does DDF use that?](#)
- [How do I parse a filter to interpret it for a provider?](#)
- [I don't see fuzzy capabilities in the OGC Filter, how is that represented?](#)
- [What are property names and which ones does DDF use?](#)
- [Can I do a nearest neighbor search in DDF?](#)

- How do I represent a relative temporal search in the OGC Filter?
- How do I represent an entry criteria search in the OGC Filter?

How can I create a filter?

First get a reference to a `FilterBuilder` from the OSGi registry. See [Working with OSGi](#) for more details on Blueprint injection.

```
<reference id="filterBuilder" interface="ddf.catalog.filter.FilterBuilder"
/>
```

Creating filters is simple with the Filter Builder API. Here is a simple example of "AND"ing a contextual search with a temporal search.

```
// filterBuilder injected via Blueprint.
ddf.catalog.filter.FilterBuilder filterBuilder;
String searchPhrase = "ied";

org.opengis.filter.Filter contextualFilter =
filterBuilder.attribute(Metacard.ANY_TEXT).is().like().text(searchPhrase);

Date endDate = Calendar.getInstance().getTime();
Date startDate = new Date(endDate.getTime() - 60000);

org.opengis.filter.Filter temporalFilter =
filterBuilder.attribute(Metacard.MODIFIED).is().during()
    .dates(temporalFilter.getStartDate(), temporalFilter.getEndDate());

org.opengis.filter.Filter finalFilter =
filterBuilder.allOf(contextualFilter, temporalFilter);
```

Compare this to using the Geotools `FilterFactoryImpl`. The Filter Builder API will create filters in the correct format for the widest support by [Sources](#).

```

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

org.opengis.filter.expression.Expression propertyName =
filterFactory.property(Metacard.ANY_TEXT) ;
String searchPhrase = "ied" ;
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

org.opengis.filter.Filter contextualFilter =
filterFactory.like(propertyName,searchPhrase, wildcardChar,singleChar,
escapeChar, isCaseSensitive);

Date endDate = Calendar.getInstance().getTime();
Date startDate = new Date(endDate.getTime() - 60000) ;

org.opengis.temporal.Instant startInstant = new
org.geotools.temporal.object.DefaultInstant(new
DefaultPosition(startDate));
org.opengis.temporal.Instant endInstant = new
org.geotools.temporal.object.DefaultInstant(new DefaultPosition(endDate));
org.opengis.temporal.Period period = new
org.geotools.temporal.object.DefaultPeriod(startInstant, endInstant);

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

org.opengis.filter.Filter finalFilter = filterFactory.and(contextualFilter,
temporalFilter) ;

```

I see there is a Geotools Query, does DDF use that?

No, DDF has its own Query interface which extends `org.opengis.filter.Filter`.

How do I parse a filter to interpret it for a provider?

The simplest approach is to use the `FilterAdapter` and `FilterDelegate`. See [Developing a Filter Delegate](#) for more details.

Alternatively, use the interface `org.opengis.filter.FilterVisitor` and "visit" each part of the Filter tree. The `FilterVisitor` uses the [Visitor pattern](#). Geotools has a few implementations of the `FilterVisitor` interface. One of note is the `DefaultFilterVisitor` which visits every node in the Filter tree. One can overwrite certain methods of this `FilterVisitor` to add one's own business logic for the various filters. For more information on parsing, see [Parsing Filters](#) section.

In all cases, the provider is responsible for handling the filter, including when the filter is null. The `OpenSearchSource` in DDF checks for a null filter and silently handles it, i.e., no exception is thrown. It is the decision of the provider as to how the filter is handled.

I don't see fuzzy capabilities in the OGC Filter, how is that represented?

In order to represent fuzzy searching, a fuzzy function was created. The fuzzy function is intended to signify when a property should be searched in a fuzzy manner. When it is not present, then no fuzzy capability is intended. More information about the fuzzy function can be found in [Creating Filters](#).

What are property names and which ones does DDF use?

As described in the [Filter Encoding Specification](#), a property is "a facet or attribute or an object referenced by name." In short, property names can be used to represent the value of a property for a particular instance of an object. An example could be title or id. The queryable properties of DDF are labeled as constants of a `Metacard` class.

Can I do a nearest neighbor search in DDF?

OGC Filter does not explicitly have a Filter for a nearest neighbor operation, but DDF does have an equivalent query. What is recommended is to use the `Beyond` filter and provide it with a property and geometry criteria with the most important part being that 0 is provided as the distance. See an example query implementation below:

```
// filterBuilder injected via Blueprint.
ddf.catalog.filter.FilterBuilder filterBuilder;

String wkt = "POINT (10 20)";

org.opengis.filter.Filter spatialFilter =
builder.attribute("location").nearestTo().wkt(wkt, 0.0);
```

How do I represent a relative temporal search in the OGC Filter?

See example temporal offset filter below:

```
// filterBuilder injected via Blueprint.
ddf.catalog.filter.FilterBuilder filterBuilder;
long offsetInMilliseconds = ...;
org.opengis.filter.Filter temporalFilter =
filterBuilder.attribute(Metacard.EFFECTIVE).is().during().last(offsetInMil
lisSeconds);
```

How do I represent an entry criteria search in the OGC Filter?

The property represented by the constant `Metacard.ID` is used to represent the id of the record to be found, so one could do a search where (`Metacard.ID = <given_id>`). See an example entry criteria search below:

```
// filterBuilder injected via Blueprint.
ddf.catalog.filter.FilterBuilder filterBuilder;
filterBuilder.attribute(Metacard.ID).is().text(id);
```

DDF 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. Below is a table of 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. Below is a table mapping Metacard types to supported comparison operators.

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

Below is a table describing each comparison operator.

Operator	Description
PropertyIsBetween	Lower <= Property <= Upper
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.

	And	Not	Or
Supported Filters	✓	✓	✓

Temporal Operators

Temporal operators compare a date associated with a property name to a given Literal date or date range. Below is a table showing supported temporal operators.

	After	AnyInteracts	Before	Begins	BegunBy	During	EndedBy	Meets	MetBy	OverlappedBy	TContains	TEquals	TOverlaps
DATE_TYPE	✓		✓			✓							

Below is a table describing 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. Below is a table showing supported spatial operators.

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

Below is a table describing 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 This is 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

Working with Transformers

Overview

Transformers are utility objects used to transform a set of standard DDF components into a desired format such as into PDF, HTML, XML, or any other format. For instance a transformer can be used to convert a set of query results into an easy-to-read HTML format ([HTML 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 though is 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.

Usage

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.

Catalog Framework

The `CatalogFramework` provides convenience 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

```
1 // inject CatalogFramework instance or retrieve an instance
2 private CatalogFramework catalogFramework;
3
4 public RSSEndpoint(CatalogFramework catalogFramework)
5 {
6     this.catalogFramework = catalogFramework ;
7     // implementation
8 }
9
10 // Other implementation details ...
11
12 private void convert(QueryResponse queryResponse ) {
13     // ...
14     String transformerId = "rss";
15
16     BinaryContent content = catalogFramework.transform(queryResponse,
17 transformerId, null);
18
19     // ...
20 }
```

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

Dependency Injection

Using Blueprint or another injection framework, transformers can be injected from the OSGi Service Registry. See [Working with OSGi](#) for more details on how to use injected instances.

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);
```

See [Working with OSGi - Service Registry](#) for more details.

OSGi Service Registry

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.getName(), "(shortname=" + transformerId + ")");  
InputTransformer inputTransformer = (InputTransformer)  
context.getService(refs[0]);  
Metacard entry = inputTransformer.transform(messageInputStream);
```

Working with Resources

Overview

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 which a client can use to know how to interpret the binary

data.

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` in the 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 under the Options heading 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.

See the [Developing a Resource Reader](#) section or the [Developing a Source](#) section in the Developer's Guide for more information and examples.

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 however 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. 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`.

Store Resource

```
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
```

See the [Developing a Resource Writer](#) section in the Developer's Guide for more information and examples.

BinaryContent

`BinaryContent` is an object used as a container to store translated or transformed DDF components. `Resource` extends `BinaryContent` and includes 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.

Commons-DDF Utilities

Explains this bundle's classes and usage.

Overview

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.

Noteworthy Classes

FuzzyFunction

`ddf.catalog.impl.filter.FuzzyFunction` class is used to signal 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.

Working with Settings

DDF provides the ability to obtain DDF Settings/Properties. For a list of DDF Settings view the Catalog API and Global Settings in the Integrator's Guide. The `DdfConfigurationWatcher` will provide an update of properties to watchers. For example, if the Port number changes, the `DDF_PORT` property value will be propagated to the watcher(s) in the form of a `Map`.

To obtain the property values follow these steps:

1. Import and Implement the `ddf.catalog.util.DdfConfigurationWatcher` Interface.

Implement DdfConfigurationWatcher

```
public class SettingsWatcher implements DdfConfigurationWatcher
```

2. Get properties map and search for the property.

Handle Properties

```
public void ddfConfigurationUpdated( Map properties )
{
    //Get property by name
    Object value = properties.get( DdfConfigurationManager.DDF_HOME_DIR
);
    if ( value != null )
    {
        this.ddfHomeDir = value.toString();
        logger.debug( "ddfHomeDir = " + this.ddfHomeDir );
    }
}
```

3. Export the Watcher Class as a Service in the OSGi Registry. The example below uses the Blueprint dependency injection framework to add this Watcher to the OSGi Registry. The `ddf.catalog.DdfConfigurationManager` will search for `ConfigurationWatcher(s)` to send properties updates.

Blueprint Example of Export

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

<!-- create the bean -->
<bean id="SettingsWatcher" class="ddf.catalog.SettingsWatcher">
  <cm:managed-properties
    persistent-id="ddf.catalog.SettingsWatcher"
    update-strategy="container-managed" />
</bean>

<!-- export the bean in the service registry as a
DdfConfigurationWatcher -->
<service ref="SettingsWatcher"
interface="ddf.catalog.util.DdfConfigurationWatcher">
</service>

</blueprint>
```

4. Import the DDF packages to the bundle's manifest for run-time (in addition to any other required packages):
`Import-Package: ddf.catalog, ddf.catalog.util, ddf.catalog.*`
5. Deploy the packaged service to DDF.
 - Check the [Working with OSGi - Bundles](#) section.

Developing an Endpoint

Prerequisites

1. Understand the desired component for development as described in the [DDF Catalog](#) section.
 - Review existing implementations in the source code and application documentation.
2. Have an IDE and the ability to create OSGi bundles.
3. Understand the [Use of the Whiteboard Design Pattern](#) section and how to publish services to the OSGi service registry.

Overview

[Endpoints](#) expose the [Catalog Framework](#) to Endpoint clients that work with metadata. Common Endpoint operations include create, update, delete, and query. SOAP and REST are two example protocols that are exposed as Endpoints. See [Included Endpoints](#) for more examples.

Creating a New Endpoint

Creating an Endpoint consists of five main steps:

1. Create a Java class that implements the Endpoint's business logic. Example: Creating a web service that external clients can invoke.
2. Add Endpoint's business logic, invoking `CatalogFramework` calls as needed.
3. Import the DDF packages to the bundle's manifest for run-time (in addition to any other required packages):

```
Import-Package: ddf.catalog, ddf.catalog.*
```

It is recommended to use the maven bundle plugin to create the Endpoint bundle's manifest as opposed to directly editing the manifest file.

4. Retrieve an instance of `CatalogFramework` from the OSGi registry.
 - Check the [Working with OSGi - Service Registry](#) section for examples.
5. Deploy the packaged service to DDF.
 - Check the [Working with OSGi - Bundles](#) section.

No implementation of an interface is required

Unlike other DDF components that require you to implement a standard interface, no implementation of an interface is required in order to create an endpoint.

Common Endpoint Business Logic

Methods	Use
Ingest	Add, modify, and remove metadata using the ingest-related <code>CatalogFramework</code> methods: <code>create</code> , <code>update</code> , and <code>delete</code> .
Query	Request metadata using the <code>query</code> method.
Source	Get available <code>Source</code> information.
Resource	Retrieve products referenced in Metacards from Sources.
Transform	Convert common Catalog Framework data types to and from other data formats.

See [Working with the Catalog Framework](#) for more details.

Developing a Catalog Plugin

Prerequisites

1. Understand the desired component for development as described in the [DDF Catalog](#) section.
 - Review existing implementations in the source code and application documentation.
2. Have an IDE and the ability to create OSGi bundles.
3. Understand the [Use of the Whiteboard Design Pattern](#) section and how to publish services to the OSGi service registry.

Overview

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.

View the [Catalog Framework section](#) of the Integrator's Guide for more information on how these plugins fit in the ingest and query flows.

Creating New Plugins

Implement Plugin Interface

The following types of plugins can be created:

Plugin Type	Plugin Interface	Description	Example
Pre-Ingest	<code>ddf.catalog.plugin.PreIngestPlugin</code>	Runs prior to the Create/Update/Delete method is sent to the <code>CatalogProvider</code>	metadata validation services
Post-Ingest	<code>ddf.catalog.plugin.PostIngestPlugin</code>	Runs after the Create/Update/Delete method being sent to the <code>CatalogProvider</code>	<code>EventProcessor</code> for processing and publishing event notifications to subscribers

Pre-Query	<code>ddf.catalog.plugin.PreQueryPlugin</code>	Runs prior to the Query/Read method being sent to the Source	An example is not included with DDF
Post-Query	<code>ddf.catalog.plugin.PostQueryPlugin</code>	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	<code>ddf.catalog.plugin.PreSubscription</code>	Runs prior to a Subscription being created or updated	Modify a query prior to creating a subscription
Pre-Delivery	<code>ddf.catalog.plugin.PreDeliveryPlugin</code>	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	<code>ddf.catalog.plugin.PreResource</code>	Runs prior to a Resource being retrieved	An example is not included with DDF
Post-Resource	<code>ddf.catalog.plugin.PostResource</code>	Runs after a Resource is retrieved, but before it is sent to the Endpoint	Verification of a resource prior to returning to a client

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 (described in the [Working with OSGi](#) section).
 - a. Import DDF packages
 - b. Register plugin class as service to OSGi registry
4. Deploy to DDF
 - Check the [Working with OSGi - Bundles](#) section.

Please view the Javadoc for more information on all Requests and Responses in the `ddf.catalog.operation` and `ddf.catalog.event` packages.

Pre-Ingest

1. Create 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 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 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;
```

An example of an implementation of a `PreQueryPlugin.process()` method's business logic is shown below and can be found in the DDF SDK in the `sample-plugins` project.

How to build PreQueryPlugin filter

```
FilterDelegate<Filter> delegate = new
CopyFilterDelegate(filterBuilder);
try {
    // Make a defensive copy of the original filter (just in case
    anyone else expects
    // it to remain unmodified)
    Filter copiedFilter = filterAdapter.adapt(query, delegate);

    // Define the extra query clause(s) to add to the copied filter
    // This will create a filter with a search phrase of:
    //      (((("schematypesearch") and ("test" and
    ({/ddms:Resource/ddms:security/@ICISM:releasableTo}: "ISAF" or
    {/ddms:Resource/ddms:security/@ICISM:releasableTo}: "CAN"))))
    Filter contextualFilter =
    filterBuilder.attribute(Metacard.ANY_TEXT).like().text("test");
    Filter releasableToFilter1 =
    filterBuilder.attribute("/ddms:Resource/ddms:security/@ICISM:releasab
    leTo").like().text("ISAF");
    Filter releasableToFilter2 =
    filterBuilder.attribute("/ddms:Resource/ddms:security/@ICISM:releasab
    leTo").like().text("CAN");
    Filter orFilter = filterBuilder.anyOf(releasableToFilter1,
    releasableToFilter2);
    Filter extraFilter = filterBuilder.allOf(contextualFilter,
    orFilter);

    // AND this PreQueryPlugin's extra query clause(s) to the copied
    filter
    Filter modifiedFilter = filterBuilder.allOf(copiedFilter,
    extraFilter);

    // Create a new QueryRequest using the modified filter and the
    attributes from the original query
    QueryImpl newQuery = new QueryImpl(modifiedFilter,
    query.getStartIndex(),
        query.getPageSize(), query.getSortBy(),
        query.requestsTotalResultsCount(), query.getTimeoutMillis());
    newQueryRequest = new QueryRequestImpl(newQuery,
    input.isEnterprise(), input.getSourceIds(), input.getProperties());
}
catch (UnsupportedQueryException e)
{
    throw new PluginExecutionException(e);
}
```

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="[[SamplePreQueryPlugin]]"  
interface="ddf.catalog.plugin.PreQueryPlugin" />
```

Post-Query

1. Create 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:

Blueprint descriptor example

```
<service ref="[[SamplePostQueryPlugin]]"  
interface="ddf.catalog.plugin.PostQueryPlugin" />
```

Pre-Delivery

1. Create 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="[[SamplePreDeliveryPlugin]]"
interface="ddf.catalog.plugin.PreDeliveryPlugin" />

```

Pre-Subscription

1. Create 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;

```

An example of an implementation of a `PreSubscriptionPlugin.process()` method's business logic is shown below and can be found in the DDF SDK in the `sample-plugins` project.

PreSubscriptionPlugin example

```
FilterDelegate<Filter> delegate = new
CopyFilterDelegate(filterBuilder);
try {
    // Make a defensive copy of the original filter (just in case
    anyone else expects
    // it to remain unmodified)
    Filter copiedFilter = filterAdapter.adapt(input, delegate);

    // Define the extra query clause(s) to add to the copied filter
    Filter extraFilter =
filterBuilder.attribute("/ddms:Resource/ddms:security/@ICISM:releasab
leTo").like().text("CAN");

    // AND the extra query clause(s) to the copied filter
    Filter modifiedFilter = filterBuilder.allOf(copiedFilter,
extraFilter);

    // Create a new subscription with the modified filter
    newSubscription = new SubscriptionImpl(modifiedFilter,
input.getDeliveryMethod(),
        input.getSourceIds(), input.isEnterprise());
}
catch (UnsupportedQueryException e)
{
    throw new PluginExecutionException(e);
}
```

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.event
```

4. Export the service to the OSGi registry:

Blueprint descriptor example

```
<service ref="[SamplePreSubscriptionPlugin]"
interface="ddf.catalog.plugin.PreSubscriptionPlugin" />
```

Pre-Resource

1. Create 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 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" />
```

Developing a Source

Prerequisites

1. Understand the desired component for development as described in the [DDF Catalog](#) section.
 - Review existing implementations in the source code and application documentation.
2. Have an IDE and the ability to create OSGi bundles.
3. Understand the [Use of the Whiteboard Design Pattern](#) section and how to publish services to the OSGi service registry.

Overview

Sources are components that enable DDF to talk to back-end services. They let DDF do 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.

1. Catalog Provider - `ddf.catalog.source.CatalogProvider`
Used to communicate with back-end storage. Allows for Query and Create/Update/Delete operations.
2. Federated Source - `ddf.catalog.source.FederatedSource`
Used to communicate with remote systems. Only allows query operations.
3. 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.

Implementing any of the source types follows a similar format:

1. Create a new class that implements the specified Source interface.
2. Implement the required methods.
3. Create OSGi descriptor file to communicate with the OSGi registry (described in the [Working with OSGi](#) section).
 - a. Import DDF packages
 - b. Register source class as service to OSGi registry
4. Deploy to DDF
 - Check the [Working with OSGi - Bundles](#) section.

Catalog Provider

1. Create 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:

- [Dummy Catalog Provider](#)

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

Federated Source

1. Create Java class that implements `FederatedSource`

```
public class TestFederatedSource implements  
ddf.catalog.source.FederatedSource
```

2. Implement the required methods of the `ddf.catalog.source.FederatedSource` interface.
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" />
```

The DDF Integrator's Guide provides details on the following Federated Sources that come with DDF out of the box:

- [OpenSearch Source](#)

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

Connected Source

1. Create Java class that implements `ConnectedSource`

```
public class TestConnectedSource implements  
ddf.catalog.source.ConnectedSource
```

2. Implement the required methods of the `ddf.catalog.source.ConnectedSource` interface.
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" />
```

Please Note

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. We have seen 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".

External Resources

[Three Rules for Effective Exception Handling](#)

Developing a Filter Delegate

Overview

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 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. See the DDF JavaDoc for more details about what is expected of each `FilterDelegate` method.

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. See [Working with OSGi](#) for more details on Blueprint injection.

```
<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
```

See [Developing a Source](#) for more details.

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
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 with in 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

Developing Transformers

Transformers provide a simple way to transform data formats for common Catalog objects such as Metacards and Query Responses. Transformers are available to all bundles through the Catalog or OSGi Service Registry. For more information on how transformers can be used see [Working with Transformers](#).

- [Developing an Input Transformer](#)
- [Developing a Metacard Transformer](#)
- [Developing a Query Response Transformer](#)
- [Developing a XSLT Transformer](#)

Developing an Input Transformer

- [Prerequisites](#)
- [Overview](#)
- [Creating an InputTransformer Using Java](#)
 - [Variable Descriptions](#)
- [Creating an Input Transformer Using Apache Camel](#)
 - [Design Pattern](#)
 - [From](#)
 - [To](#)
 - [Message Formats](#)
 - [InputTransformer](#)

- [Examples](#)
- [Additional Reading](#)

Prerequisites

1. Understand the desired component for development as described in the [DDF Catalog](#) section.
 - Review existing implementations in the source code and application documentation.
2. Have an IDE and the ability to create OSGi bundles.
3. Understand the [Use of the Whiteboard Design Pattern](#) section and how to publish services to the OSGi service registry.

Overview

An `InputTransformer` is used to transform data from an `InputStream` into a `Metacard`. Input Transformers can be requested from the OSGi Service Registry by Endpoints or any other bundle. See [Included Input Transformers](#) for examples.

Creating an InputTransformer 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>
...
```

5. Deploy OSGi Bundle to OSGi runtime.

Variable Descriptions

Blueprint Service properties

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

Creating an Input Transformer Using Apache Camel

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

Design Pattern

From

When using **from** `catalog:inputtransformer?id=text/xml` then 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` then an [Input Transformer](#) with an id matching `text/xml` will be discovered from the OSGi registry and invoked.

Message Formats

InputTransformer

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

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=text/xml;id=ddms20)" />
    </route>
  </camelContext>
</blueprint>
```

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
3	Defines start of an Apache Camel route
4	<p>Defines the endpoint/consumer for the route. In this case it is the DDF custom catalog component that is an <code>InputTransformer</code> registered with</p> <p>an id of <code>text/xml;id=vehicle</code> meaning it can transform an <code>InputStream</code> of vehicle data into a Metacard.</p> <p>Note that the specified XSL stylesheet must be on the classpath of the bundle that this blueprint file is packaged in.</p>
5	Defines the XSLT to be used to transform the vehicle input into DDMS format using the Apache Camel provided XSLT component.
6	<p>Defines the route node that accepts DDMS formatted input and transforms it into a Metacard, using the DDF custom catalog component</p> <p>that is an <code>InputTransformer</code> registered with an id of <code>text/xml;id=ddms20</code></p>

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`

Additional Reading

- [Working with Transformers](#)
- [BinaryContent](#)

Developing a Metacard Transformer

Prerequisites

1. Understand the desired component for development as described in the [DDF Catalog](#) section.
 - Review existing implementations in the source code and application documentation.
2. Have an IDE and the ability to create OSGi bundles.
3. Understand the [Use of the Whiteboard Design Pattern](#) section and how to publish services to the OSGi service registry.

Overview

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. See [Included Metacard Transformers](#) for examples.

Creating 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>
...
```

5. Deploy OSGi Bundle to OSGi runtime.

Variable Descriptions

Blueprint Service properties

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

Additional Reading

- [Working with Transformers](#)
- [BinaryContent](#)

Developing a Query Response Transformer

Prerequisites

1. Understand the desired component for development as described in the [DDF Catalog](#) section.
 - Review existing implementations in the source code and application documentation.
2. Have an IDE and the ability to create OSGi bundles.
3. Understand the [Use of the Whiteboard Design Pattern](#) section and how to publish services to the OSGi service registry.

Overview

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. See [Included Query Response Transformers](#) for examples.

Creating 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>
...
```

5. Deploy OSGi Bundle to OSGi runtime.

Variable Descriptions

Blueprint Service properties

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

Additional Reading

- [Working with Transformers](#)
- [BinaryContent](#)

Developing a XSLT Transformer

Prerequisites

1. Understand the desired component for development as described in the [DDF Catalog](#) section.
 - Review existing implementations in the source code and application documentation.
2. Have an IDE and the ability to create OSGi bundles.
3. Understand the [Use of the Whiteboard Design Pattern](#) section and how to publish services to the OSGi service registry.

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.

Implement 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>${project.artifactId}</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 Response 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

Create a Maven project named `service-rss-transformer`. 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>${project.artifactId}</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.

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. 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:ddms="http://metadata.dod.mil/mdr/ns/DDMS/2.0/"
  xmlns:gml="http://www.opengis.net/gml" exclude-result-prefixes="xsl ddms
```

```
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: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.

Developing a Resource Reader

Overview

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 `FileResourceReader`. 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. See the [Working with Resources](#) section of the Developer's Guide for more information.

Describes how to create a `ResourceReader`.

Prerequisites

1. Understand the desired component for development as described in the [DDF Catalog](#) section.
 - Review existing implementations in the source code and application documentation.
2. Have an IDE and the ability to create OSGi bundles.
3. Understand the [Use of the Whiteboard Design Pattern](#) section and how to publish services to the OSGi service registry.

Creating A New ResourceReader

Creating a `ResourceReader` consists of 2 main steps:

1. Create a Java class that implements the `ddf.catalog.resource.ResourceReader` interface.
2. Deploy the OSGi bundled packaged service to the DDFrun-time.
 - Check the [Working with OSGi - Bundles](#) section.

Implementing the ResourceReader interface

```

public class TestResourceReader implements
ddf.catalog.resource.ResourceReader

```

`ResourceReader` has a couple key methods where most of the work is performed.

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 here: <http://www.w3.org/Addressing/URL/uri-spec.html>.

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`.

The "Map<String, Serializable> arguments" parameter is passed in to support any options or additional information associated with retrieving the resource.

Implementing `retrieveResource()`

1. Define supported schemes (e.g. file, http, etc)
2. Check if the incoming `URI` matches a supported scheme, if not throw `ResourceNotSupportedException`.
3. For example:

```
if ( !uri.getScheme().equals("http") )
{
    throw new ResourceNotSupportedException("Unsupported scheme
received, was expecting http")
}
```

4. Implement the business logic.
5. 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();
```

The `Resource` needs to be accessible from the DDF installation. 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)

6. Return `Resource` in `ResourceResponse`
7. For example:

```
return ResourceResponseImpl( new ResourceImpl( new
BufferedInputStream( is ), new MimeType( mimeType ), url.getFile() )
);
```

8. 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 / should be passed. For single-use `ResourceReaders` (like a `URLResourceReader`) there may only be one scheme that it can accept while for others they 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.

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 by looking at 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" />
```

Developing a Resource Writer

Overview

Before implementing a Resource Writer see the Content Framework for alternatives.

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`. See the [Working with Resources](#) section in the Developer's guide for more information.

Prerequisites

1. Understand the desired component for development as described in the [DDF Catalog](#) section.
 - Review existing implementations in the source code and application documentation.
2. Have an IDE and the ability to create OSGi bundles.
3. Understand the [Use of the Whiteboard Design Pattern](#) section and how to publish services to the OSGi service registry.

Creating a New ResourceWriter

Creating a ResourceWriter consists of these main steps:

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;
    }
}
```

2. Register the implementation as a Service in the OSGi Service Registry.

Blueprint Service Registration Example

```
...
<service ref="[[ResourceWriterReference]]"
interface="ddf.catalog.resource.ResourceWriter" />
...
```

3. Deploy the OSGi bundled packaged service to the DDF run-time.
 - Check the [Working with OSGi - Bundles](#) section.

See the DDF Catalog API Javadoc for more information as to the methods required in implementing the interface.

Developing a Registry Client

Description

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`.

Developing at the Framework Level

- [Developing Complementary Frameworks](#)
- [Developing Console Commands](#)

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. Reference 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

Developing Console Commands

Console Commands

DDF supports development of custom console commands. For more information, see the Karaf website on [Extending the Console](#).

Custom DDF Console Commands

DDF includes custom commands for working with the Catalog, as described in the [Using Console Commands](#) section.

Developing Action Components

Action Framework

Description

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 though 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 care how the link is generated, but mostly that the link will work in 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: (1) it allows the feature to be independent of implementations, and (2) 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`

Usage

If wanting to provide a service such as a link to a record, one would implement the `ActionProvider` interface. An `ActionProvider` essentially provides an `Action` 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 provides 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 `null`. 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 mostly a URL that when invoked provides a resource or executes intended business logic.

Taxonomy

An Action Provider registers an `id` as a service property in the OSGi 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
<code>catalog.data.metacard.view</code>	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 or JSON, etc.
<code>catalog.data.metacard.thumbnail</code>	Provides a valid URL to the bytes of a thumbnail (Metacard.THUMBNAIL) with MIME type <code>image/jpeg</code>
<code>catalog.data.metacard.html</code>	Provides a valid URL that when invoked provides an HTML representation of the Metacard
<code>catalog.data.metacard.resource</code>	Provides a valid URL that when invoked provides the underlying resource of the Metacard
<code>catalog.data.metacard.metadata</code>	Provides a valid URL to the XML Metadata in the Metacard (Metacard.METADATA)

Developing for the DDF Marketplace

Overview

The DDF Marketplace provides a browser based interface to applications that can be downloaded and installed to the DDF Framework. These are

applications that are in addition to the DDF "out-of-the-box" components.

The DDF components are packaged as Karaf features, which are collections of OSGi bundles. These features can be installed/uninstalled using the DDF Admin Console or command line console. DDF Marketplace applications also consist of one or more OSGi bundles, and possibly supplemental external files. These Marketplace application are packaged as Karaf kar files for easy download and installation.

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

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 (available as of Karaf v2.2.5, which DDF 2.X is based upon). This goal reads all of the features specified in the features 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.

An example of using the `create-kar` goal is shown below:

create-kar goal

```
<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>${project.build.directory}/doesNotExist</resourcesDir>

        <!--
Location of the features.xml file. If it references
properties that need to be filtered, e.g., ${project.version}, it will need
to be

        filtered by the maven-resources-plugin.
-->

<featuresFile>${basedir}/target/classes/features.xml</featuresFile>

        <!-- Name of the kar file (.kar extension added by
default). If not specified, defaults to ${project.build.finalName} -->
        <finalName>ddf-isis-${project.version}</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 used when creating a listing in the DDF Marketplace for the application.

Including Data Files in a KAR File

Sometimes the 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 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

Installing a KAR File

When the user downloads a Marketplace application by clicking on the Installation link, the application's kar file is downloaded. This kar file should 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. The user can then go to the Admin Console's Features tab and verify the new feature(s) is installed.

Installing a KAR File for DDF 2.0.0 Release

DDF 2.0.0 is based on Karaf v2.2.2, which has a bug in the KAR deployer. Therefore a more manual approach to install a kar file is required rather than just dropping the kar file in the `<DDF_INSTALL_DIR>/deploy` directory.

After downloading the Marketplace application's kar file:

1. Copy the kar file to a temp directory
2. Unjar the kar file in this temp directory
3. Edit the `etc/org.ops4j.pax.url.mvn.cfg` configuration file. Uncomment the `org.ops4j.pax.url.mvn.repositories` property if it is commented out.
Append the `org.ops4j.pax.url.mvn.repositories` property with `file:<temp_dir>/<kar_root_dir>/repository`
4. Restart DDF
5. Add the kar's feature repository URL, either using the command line console or Features tab in the Admin Console

Example:

Assume the Marketplace application's kar file is named `new-app-1.2.3.kar`

1. Copy `new-app-1.2.3.kar` to `/my_kars` temp directory
2. `unjar xvf /my_kars/new-app-1.2.3.kar`
3. Edit the `<DDF_INSTALL_DIR>/etc/org.ops4j.pax.url.mvn.cfg` configuration file and append `file:/my_kars/new-app-1.2.3/repository` to the `org.ops4j.pax.url.mvn.repositories` property
4. Restart DDF
5. At the `ddf@local>` prompt enter `features:addurl`
`file:/my_kars/new-app-1.2.3/repository/new-app/1.2.3/new-app-1.2.3-features.xml`
6. At the `ddf@local>` prompt enter `features:list` and the new feature(s) loaded by the kar file should be displayed and active

Developing Security Token Service Components

Developing Token Validators

Description

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.

Usage

At the moment, validators must be added to the cxf-sts.xml blueprint file manually. There is a section labeled as the "Delegate Configuration" in that file. That is where the validator must be added along with the existing validators. In the future, we expect this to be pluggable without changes to the blueprint file.

Update: The validator services that are currently bundled on their own are webSSOTokenValidator, SamlTokenValidator, and x509TokenValidator. Each has its own blueprint.xml file that defines and exports the service. They can be individually turned on/off as a bundle/service.

Building

Prerequisites

It is required to have installed Maven 3.0 or higher on the machine that will be attempting to build the latest release.

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, see the Maven [settings reference page](#).

Sample settings.xml file

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

Handy Tip on Encrypting Passwords

See this Maven [guide](#) on how to encrypt the passwords in your settings.xml.

Procedures

Run the Build

1. After the settings.xml has been properly configured, run the build from the root-level project in a command line prompt or terminal using the following command

```
mvn clean install
```

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

Build times may vary based on network speed and machine specifications.

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 causing the heap space to run out in the main maven JVM. To fix this issue, set the system

variable MAVEN_OPTS with the value -Xmx512m -XX:MaxPermSize=128m before running the build.

Example on Linux system with the bash shell: export MAVEN_OPTS='-Xmx512m -XX:MaxPermSize=128m'

DDF Camel Components

Components Included

DDF includes the following Component implementations via URIs.

Component / ArtifactId / URI	Description
<div>Catalog / catalog-core-camelcomponent<div>catalog:framework</div></div>	Exposes the Catalog Framework .

Catalog Framework Camel Component

Overview

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 that you would want to delete. 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>

```

Javadocs

Released APIs

The following are APIs and specifications that will be released for developers to implement or use as direction.

Catalog Core API

[zip](#)
[html](#)

Beta APIs

The following are APIs being improved and are considered to be in BETA status. They are available for review. Beta APIs may change in the future with no guarantee of backwards compatibility until the APIs have been released.

Content Core API

Action Core API

Mime Core API

Security Core API

Security Encryption API

[zip](#)
[zip](#)
[zip](#)
[zip](#)
[html](#)
[html](#)
[html](#)
[html](#)

Private APIs

The Metrics Framework and Metrics Endpoint are for internal use at this time. The endpoint is likely to change in future versions, hence any custom applications built to make use of it, should be done with caution.

Quick Start

DDF in 5 Minutes

This quick tutorial will demonstrate:

- ☒ Installation
- ☒ Catalog Capabilities: Ingest and query using every endpoint
- ☒ Use of the Content Framework
- ☒ Metrics Reporting

Installation

- Install DDF by unzipping the zip file (this installation directory will henceforth be called `DISTRIBUTION_INSTALL_DIR`)
- Start DDF by running the `<DISTRIBUTION_INSTALL_DIR>/bin/ddf script` (or `ddf.bat` on Windows)
- Verify distribution is up by:
 - executing `list` command in command line console - all bundles should be Active (or Resolved for fragments)
 - go to <http://localhost:8181/services> and verify 3 REST services are available: metrics, catalog, and catalog/query

From this `services` URL, click on the links to each REST service's WADL to see its interface.

Default Catalog

The embedded Solr Catalog Provider is installed by default out-of-the-box as of DDF 2.2.0+.

Currently all services/endpoints in DDF are REST-based. There are no SOAP endpoints.

- In Web Admin Console at <http://localhost:8181/system/console/configMgr> (username:admin password:admin) configure the System Settings:
 - select Platform Global Configuration
 - enter the host and port of where distribution is running

Catalog Capabilities

- Create an entry in the Catalog by ingesting a [valid GeoJson file](#) (attached to this page).

This ingest can be done using:

- a REST client, such as Google Chrome's Advanced REST Client
- OR
- by using the following curl command to POST to the Catalog REST CRUD endpoint

Windows Example

```
curl.exe -H "Content-type: application/json;id=geojson" -i -X
POST -d @"C:\path\to\geojson_valid.json"
http://localhost:8181/services/catalog
```

*NIX Example

```
curl -H "Content-type: application/json;id=geojson" -i -X POST -d
@geojson_valid.json http://localhost:8181/services/catalog
```

where:

-H adds a HTTP header. In this case Content-type header `application/json;id=geojson` is added to match the data being sent in the request

-i requests that HTTP headers are displayed in the response

-X specifies the type of HTTP operation. For this example, it is necessary to POST (ingest) data to the server.

-d specifies the data sent in the POST request. The @ character is necessary to specify that the data is a file.

The last parameter is the server's URL to send the data to.

This should return a response similar to the following (the actual catalog ID in the id and Location URL fields will be different):

Sample Response

```
HTTP/1.1 201 Created
Content-Length: 0
Date: Mon, 22 Apr 2013 22:02:22 GMT
id: 44dc84da101c4f9d9f751e38d9c4d97b
Location:
http://localhost:8181/services/catalog/44dc84da101c4f9d9f751e38d
9c4d97b
Server: Jetty(7.5.4.v20111024)
```

- Verify the entry was successfully ingested by entering in a browser the URL returned in the POST response's HTTP header. For instance in our example, it was `http://localhost:8181/services/catalog/44dc84da101c4f9d9f751e38d9c4d97b`

This should display the catalog entry in XML within the browser.

- Verify catalog entry exists by executing a query via the OpenSearch endpoint.
 - Enter following URL in a browser <http://localhost:8181/services/catalog/query?q=ddf>
A single result, in Atom format, should be returned.
- Verify catalog entry exists by executing a query in the SearchUI (which is OpenSearch based):
 - Point a browser to <http://localhost:8181/search>
 - A single result should be displayed in the SearchUI's results.

Use of the Content Framework

- Using the Content Framework's Directory Monitor, ingest a file so that it is stored in the Content repository with a Metacard created and inserted into the Catalog.
 - In the [Web Admin Console](#), select the Configuration tab, then select the `Content Directory Monitor` and set the Directory Path to `inbox`, and click Save.
 - Copy the attached [geojson_valid.json](#) file to the `<DISTRIBUTION_INSTALL_DIR>/inbox` directory
 - The Content Framework will:
 - ingest the file
 - store it in the content repository at `<DISTRIBUTION_INSTALL_DIR>/content/store/<GUID>/geojson_valid.json`
 - lookup the GeoJson Input Transformer based on the mime type of the ingested file
 - create a Metacard based on the metadata parsed from the ingested GeoJson file
 - insert the Metacard into the Catalog using the `CatalogFramework`
- Verify GeoJson file was stored using the Content REST Endpoint

- Install the feature `content-rest-endpoint` using the Features tab in the Admin console
- Send a `GET` command to read the content from the content repository using the Content REST Endpoint. This can be done using `curl` command below. Note that the GUID will be different for each ingest - the GUID can be determined by going to the `<DISTRIBUTION_INSTALL_DIR>/content/store` directory and copying the sub-directory in this folder (there should only be one), which is the GUID.

*NIX Example

```
curl -X GET
http://localhost:8181/services/content/c90147bf86294d46a9d35ebbd
44992c5
```

The response to the `GET` command will be the contents of the `geojson_valid.json` file originally ingested.

Metrics Reporting

- Now that several queries have been executed, go the Metrics tab in the Admin console (<http://localhost:8181/system/console/metrics>)
 - Click on the `PNG` link for the `Catalog Queries` row under the column labeled `1d`, which stands for one day. A graph of the catalog queries executed in the last 24 hours will be displayed.
 - Click the back button of the browser to return to the Metrics tab.
 - Click the `XLS` link for the `Catalog Queries` row under the column labeled `1d`.

Handy Tip


Based on the browser's configuration, either the `.xls` file will be downloaded or automatically displayed in Excel.

Release Notes

- [2.2.0.RC2](#)
- [2.2.0.RC3](#)
- [2.2.0.RC4](#)

2.2.0.RC2

Type	Key	Summary	Assignee	Reporter	Priority	Status	Resolution	Created	Updated	Due
------	-----	---------	----------	----------	----------	--------	------------	---------	---------	-----

 JIRA project doesn't exist or you don't have permission to view it.

[View these issues in JIRA](#)


2.2.0.RC3

Release Notes

- Added a separate log file that exclusively handles failed ingest attempts. Details can be found here <https://tools.codice.org/wiki/display/DDF/Catalog+Framework> under the Error Handling section.
- A new naming convention was applied to the configuration names in the webconsole.
- Simple UI can now be used to access data from a secure DDF.
- Numerous bug fixes listed below.

Resolved Issues

Type	Key	Summary	Assignee	Reporter	Priority	Status	Resolution	Created	Updated	Due
------	-----	---------	----------	----------	----------	--------	------------	---------	---------	-----

 JIRA project doesn't exist or you don't have permission to view it.
[View these issues in JIRA](#)


2.2.0.RC4

Release Notes

- Numerous bug fixes listed below.

Resolved Issues

Type	Key	Summary	Assignee	Reporter	Priority	Status	Resolution	Created	Updated	Due
------	-----	---------	----------	----------	----------	--------	------------	---------	---------	-----

 JIRA project doesn't exist or you don't have permission to view it.
[View these issues in JIRA](#)