Developing DDF Components: Developer's Guide

Table of Contents

License . . . 5
1. Developing DDF Components . . . 6
1.1. Developing Complementary Catalog Frameworks . . . 6
1.1.1. Simple Catalog API Implementations . . . 6
1.1.2. Use of the Whiteboard Design Pattern . . . 6
1.1.3. Recommendations for Framework Development . . . 6
1.1.4. Catalog Framework Reference . . . 7
1.1.4.1. Methods . . . 7
1.1.4.1.1. Create, Update, and Delete Methods . . . 7
1.1.4.1.2. Query Methods . . . 7
1.1.4.1.3. Resource Methods . . . 8
1.1.4.1.4. Source Methods . . . 8
1.1.4.1.5. Transform Methods . . . 8
1.1.4.2. Implementing Catalog Methods . . . 9
1.1.4.3. Dependency Injection . . . 9
1.1.4.4. OSGi Service Registry . . . 10
1.2. Developing Metacard Types . . . 10
1.2.1. Metacard Type Definition File . . . 10
1.3. Developing Metacard Validators . . . 12
1.3.1. Metacard Validator Definition . . . 13
1.4. Developing Global Attribute Validators . . . 14
1.4.1. Global Attribute Validators File . . . 14
1.5. Developing Attribute Types . . . 17
1.5.1. Attribute Type Definition File . . . 17
1.6. Developing Default Attribute Types . . . 19
1.6.1. Default Attribute Values . . . 19
1.7. Developing Attribute Injections . . . 20
1.7.1. Attribute Injection Definition . . . 21
1.8. Developing Endpoints . . . 23
1.9. Developing Input Transformers . . . 24
1.9.1. Create an XML Input Transformer using SaxEventHandlers . . . 25

1.9.2. Create an Input Transformer Using Apache Camel . . . 26
1.9.2.1. Input Transformer Design Pattern (Camel) . . . 26
1.9.3. Input Transformer Boot Service Flag . . . 28
1.10. Developing Metacard Transformers . . . 28
1.10.1. Creating a New Metacard Transformer . . . 28
1.11. Developing Query Response Transformers . . . 29
1.12. Developing Sources . . . 30
1.12.1. Implement a Source Interface . . . 31
1.12.1.1. Developing Catalog Providers . . . 32
1.12.1.2. Developing Federated Sources . . . 32
1.12.1.3. Developing Connected Sources . . . 33
1.12.1.4. Exception Handling . . . 33
1.12.1.4.1. Exception Examples . . . 33
1.12.1.4.2. External Resources for Developing Sources . . . 34
1.13. Developing Catalog Plugins . . . 34
1.13.1. Implementing Catalog Plugins . . . 35
1.13.1.1. Catalog Plugin Failure Behavior . . . 36
1.13.1.2. Implementing Pre-Ingest Plugins . . . 36
1.13.1.3. Implementing Post-Ingest Plugins . . . 36
1.13.1.4. Implementing Pre-Query Plugins . . . 37
1.13.1.5. Implementing Post-Query Plugins . . . 37
1.13.1.6. Implementing Pre-Delivery Plugins . . . 38
1.13.1.7. Implementing Pre-Subscription Plugins . . . 38
1.13.1.8. Implementing Pre-Resource Plugins . . . 38
1.13.1.9. Implementing Post-Resource Plugins . . . 39
1.13.1.10. Implementing Policy Plugins . . . 39
1.13.1.11. Implementing Access Plugins . . . 40
1.14. Developing Token Validators . . . 40
1.15. Developing STS Claims Handlers . . . 41
1.15.1. Example Requests and Responses for SAML Assertions . . . 48
1.15.2. BinarySecurityToken (CAS) SAML Security Token Samples . . . 48
1.15.3. UsernameToken Bearer SAML Security Token Sample . . . 55
1.15.4. X.509 PublicKey SAML Security Token Sample . . . 61
1.16. Developing Registry Clients . . . 68
1.17. Developing Resource Readers . . . 68
1.17.1. Creating a New `ResourceReader` . . . 69
1.17.1.1. Implementing the `ResourceReader` Interface . . . 69
1.17.1.2. retrieveResource . . . 69

1.17.1.3. Implement `retrieveResource()` . . . 69
1.17.1.4. `getSupportedSchemes` . . . 70
1.17.1.5. Export to OSGi Service Registry . . . 71
1.18. Developing Resource Writers . . . 71
1.18.1. Create a New `ResourceWriter` . . . 71
1.19. Developing Filters . . . 72
1.19.1. Units of Measure . . . 73
1.19.2. Filter Examples . . . 73
1.19.2.1. Contextual Searches . . . 74
1.19.2.1.1. Tree View of Creating Filters . . . 75
1.19.2.1.2. XML View of Creating Filters . . . 75
1.19.2.2. Fuzzy Operations . . . 76
1.19.3. Parsing Filters . . . 77
1.19.3.1. Interpreting a Filter to Create SQL . . . 77
1.19.3.2. Interpreting a Filter to Create XQuery . . . 78
1.19.3.2.1. FilterAdapter/Delegate Process for Figure Parsing . . . 78
1.19.3.2.2. FilterVisitor Process for Figure Parsing . . . 79
1.19.4. Filter Profile . . . 79
1.19.4.1. Role of the OGC Filter . . . 79
1.19.4.2. Catalog Filter Profile . . . 80
1.19.4.2.1. Comparison Operators . . . 80
1.19.4.2.2. Logical Operators . . . 82
1.19.4.2.3. Temporal Operators . . . 82
1.19.4.2.4. Spatial Operators . . . 82
1.20. Developing Filter Delegates . . . 83
1.20.1. Creating a New Filter Delegate . . . 83
1.20.1.1. Implementing the Filter Delegate . . . 84
1.20.1.2. Throwing Exceptions . . . 84
1.20.1.3. Using the Filter Adapter . . . 84
1.20.1.4. Filter Support . . . 85
1.21. Developing Query Options . . . 86
1.21.1. Evaluating a query . . . 87
1.21.2. Commons-DDF Utilities . . . 87
1.21.2.1. FuzzyFunction . . . 87
1.21.2.2. XPathHelper . . . 87
1.22. Developing Action Components . . . 88
1.22.1. Action Component Naming Convention . . . 88
1.22.1.1. Action Component Taxonomy . . . 88

1.23. Configuring Managed Service Factory Bundles . . . 89
1.23.1. File Format . . . 90
1.24. Developing XACML Policies . . . 92
1.24.1. XACML Policy Attributes . . . 93
1.24.2. XACML Policy Subject . . . 93
1.24.3. XACML Policy Resource . . . 93
1.24.4. Using a XACML Policy . . . 93
1.25. Assuring Authenticity of Bundles and Applications . . . 93
1.25.1. Prerequisites . . . 93
1.25.2. Signing a JAR/KAR . . . 94
1.25.2.1. Verifying a JAR/KAR . . . 94
1.26. WFS Services . . . 95
1.27. JSON Definition Files . . . 97
1.27.1. Definition File Format . . . 97
1.27.2. Deploying Definition Files . . . 97
1.28. Developing Subscriptions . . . 97
1.28.1. Subscription Lifecycle . . . 98
1.28.1.1. Creation . . . 98
1.28.1.2. Evaluation . . . 98
1.28.1.3. Update Evaluation . . . 98
1.28.1.4. Durability . . . 98
1.28.2. Creating a Subscription . . . 98
1.28.2.1. Event Processing and Notification . . . 99
1.28.2.1.1. Using DDF Implementation . . . 99
1.28.2.2. Delivery Method . . . 100
1.29. Contributing to Documentation . . . 100
1.29.1. Editing Existing Documentation . . . 101
1.29.2. Adding New Documentation Content . . . 102
1.29.3. Creating a New Documentation Template . . . 102
1.29.4. Extending Documentation in Downstream Distributions . . . 102
2. Development Guidelines . . . 103
2.1. Contributing . . . 103
2.2. OSGi Basics . . . 103
2.2.1. Packaging Capabilities as Bundles . . . 104
2.2.1.1. Creating a Bundle . . . 104
2.2.1.1.1. Bundle Development Recommendations . . . 104
2.2.1.1.2. Maven Bundle Plugin . . . 105
2.2.1.2. Third Party and Utility Bundles . . . 106

2.2.1.3. Deploying a Bundle . . . 106
2.2.1.4. Verifying Bundle State . . . 106
2.3. High Availability Guidance . . . 107
Index . . . 108

License

Copyright (c) Codice Foundation.
This work is licensed under a Creative Commons Attribution 4.0 International License.

This document last updated: 2026-10-12.

1. Developing DDF Components

Create custom implementations of DDF components.

1.1. Developing Complementary Catalog Frameworks

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

1.1.1. Simple Catalog API Implementations

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

- First, they provide a good starting point for other developers to extend functionality in the framework. For instance, extending the `MetacardImpl` allows developers to focus less on the inner workings of DDF and more on the developer's intended purposes and objectives.
- Second, the Catalog API Implementations display the proper usage of an interface and an interface's intentions. Also, they are good code examples for future implementations. If a developer does not want to extend the simple implementations, the developer can at least have a working code reference on which to base future development.

1.1.2. Use of the Whiteboard Design Pattern

The 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.

1.1.3. Recommendations for Framework Development

- Provide extensibility similar to that of the Catalog.
  - Provide a stable API with interfaces and simple implementations (refer to `http://www.ibm.com/developerworks/websphere/techjournal/1007_charters/1007_charters.html`).
- Make use of the Catalog wherever possible to store, search, and transform information.
- Utilize OSGi standards wherever possible.

    - ConfigurationAdmin
    - MetaType
- Utilize the sub-frameworks available in DDF.
    - Karaf
    - CXF
    - PAX Web and Jetty

1.1.4. Catalog Framework Reference

The Catalog Framework can be requested from the OSGi Service Registry.

Blueprint Service Reference

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

1.1.4.1. Methods

The `CatalogFramework` provides convenient methods to transform `Metacards` and `QueryResponses` using a reference to the `CatalogFramework`.

1.1.4.1.1. Create, Update, and Delete Methods

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

Example Create, Update, Delete Methods

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

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

1.1.4.1.2. Query Methods

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.

Example Query Methods

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

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

1.1.4.1.3. Resource Methods

Resource methods retrieve data resources from Sources.

Example Resource Methods

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

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

1.1.4.1.4. Source Methods

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

Example Source Methods

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

1.1.4.1.5. Transform Methods

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;
```

1.1.4.2. Implementing Catalog Methods

Query Response Transform Example

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

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

 // Other implementation details ...

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

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

    // ...

}
```

1.1.4.3. Dependency Injection

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

Blueprint Service Reference

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

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

Input Transformer Example

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

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

Metacard Transformer Example

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

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

Query Response Transformer Example

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

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

1.1.4.4. OSGi Service Registry

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

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

OSGi Service Registry Reference Example

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

1.2. Developing Metacard Types

Create custom Metacard types with Metacard Type definition files.

1.2.1. Metacard Type Definition File

To define Metacard Types, the definition file must have a `metacardTypes` key in the root object.

```
{
```

```
    "metacardTypes": [...]
}
```

The value of `metacardTypes` must be an array of Metacard Type Objects, which are composed of the `type` (required), `extendsTypes` (optional), and `attributes` (optional) keys.

Sample Top Level metacardTypes Definition

```
{
    "metacardTypes": [
        {
            "type": "my-metacard-type",
            "extendsTypes": ["core", "security"],
            "attributes": {...}
        }
    ]
}
```

The value of the `type` key is the name of the metacard type being defined. **This field is required.**

The value of the `extendsTypes` key is an array of metacard type names (strings) whose attributes you wish to include in your type. Valid Metacard Types already defined in the system or any Metacard Types already defined in this file work. Please note this section is evaluated from top to bottom so order any types used in other definitions above where they are used in the `extendsTypes` of other definitions. This key and value may be completely omitted to not extend any types.

The value of the `attributes` key is a map where each key is the name of an attribute type to include in this metacard type and each value is a map with a single key named `required` and a boolean value. Required attributes are used for metacard validation - metacards that lack required attributes are flagged with validation errors. `attributes` may be completely omitted. `required` may be omitted.

Sample Complete metacardTypes Definition

```
{
    "metacardTypes": [
        {
            "type": "my-metacard-type",
            "attributes": {
                "resolution": {
                    "required": true
                },
                "target-areas": {
                    "required": false
                },
                "expiration": {},
                "point-of-contact": {
```

```
                    "required": true
                }
            }
        }
    ]
}
```

NOTE	The DDF basic metacard attribute types are added to custom metacard types by default. If any attribute types are required by a metacard type, just include them in the `attributes` map and set `required` to `true`, as shown in the above example with `point-of-contact`.

Multiple Metacard Types in a Single File

```
{
    "metacardTypes": [
        {
            "type": "my-metacard-type",
            "attributes": {
                "resolution": {
                    "required": true
                },
                "target-areas": {
                    "required": false
                }
            }
        },
        {
            "type": "another-metacard-type",
            "attributes": {
                "effective": {
                    "required": true
                },
                "resolution": {
                    "required": false
                }
            }
        }
    ]
}
```

1.3. Developing Metacard Validators

1.3.1. Metacard Validator Definition

Metacard Validator definitions are similar to the Validators definitions. To define Metacard Validators, your definition file must have a `metacardvalidators` key in the root object.

```
{
    "metacardvalidators": {...}
}
```

The value of `metacardvalidators` is a list of maps, where each map contains a key that is the name of the metacard type the corresponding validators apply to. Its value is a list of maps of the validator configuration parameters that specify which validators to configure for the given metacard type.

```
{
    "metacardvalidators": [{
        "metacard.type" : [{
            "validator": "requiredattributes",
            "key": "value"
        }]
      }]
}
```

IMPORTANT	The configuration for the metacard validator must always contain a `validator` key that specifies the type of validator to create. Additional key/value pairs may be required to configure the validator.

The `validator` key must have a value of one of the following:

- `requiredattributes` (checks that metacards of the specified type contain certain attributes)
  - Must add the key `requiredattributes` whose value is a list of strings naming the required attributes

Examples:

```
{
  "metacardvalidators": [{
    "fallback.common": [{
        "validator": "requiredattributes",
        "requiredattributes": [
          "id",
          "title",
          "attr"
        ]
    }]
```

```
  }]
}
```

1.4. Developing Global Attribute Validators

1.4.1. Global Attribute Validators File

To define Validators, the definition file must have a `validators` key in the root object.

```
{
    "validators": {...}
}
```

The value of `validators` is a map of the attribute name to a list of validators for that attribute.

```
{
    "validators": {
        "point-of-contact": [...]
    }
}
```

Each object in the list of validators is the validator name and list of arguments for that validator.

```
{
    "validators": {
        "point-of-contact": [
            {
                "validator": "pattern",
                "arguments": [".*regex.+\\s"]
            }
        ]
    }
}
```

WARNING	The value of the `arguments` key must always be an array of strings, even for numeric arguments, for example, `["1", "10"]`

The `validator` key must have a value of one of the following:

1. `validator` Possible Values

- `size` (validates the size of Strings, Arrays, Collections, and Maps)

    - `arguments`: (2) [integer: lower bound (inclusive), integer: upper bound (inclusive)]
        - lower bound must be greater than or equal to zero and the upper bound must be greater than or equal to the lower bound
- `pattern`
    - `arguments`: (1) [regular expression]
- `pastdate`
    - `arguments`: (0) [NO ARGUMENTS]
- `futuredate`
    - `arguments`: (0) [NO ARGUMENTS]
- `range`
    - (2) [number (decimal or integer): inclusive lower bound, number (decimal or integer): inclusive upper bound]
        - uses a default epsilon of 1E-6 on either side of the range to account for floating point representation inaccuracies
    - (3) [number (decimal or integer): inclusive lower bound, number (decimal or integer): inclusive upper bound, decimal number: epsilon (the maximum tolerable error on either side of the range)]
- `enumeration`
    - `arguments`: (unlimited) [list of strings: each argument is one case-sensitive, valid enumeration value]
- `relationship`
    - `arguments`: (4+) [attribute value or null, one of mustHave|cannotHave|canOnlyHave, target attribute name, null or target attribute values as additional arguments]
- `match_any`
    - `validators`: (unlimited) [list of previously defined validators: valid if any validator succeeds]

Example Validator Definition

```
{
    "validators": {
        "title": [
            {
                "validator": "size",
                "arguments": ["1", "50"]
            },
            {
                "validator": "pattern",
```

```
                "arguments": ["\\D+"]
            }
        ],
        "created": [
            {
                "validator": "pastdate",
                "arguments": []
            }
        ],
        "expiration": [
            {
                "validator": "futuredate",
                "arguments": []
            }
        ],
        "page-count": [
            {
                "validator": "range",
                "arguments": ["1", "500"]
            }
        ],
        "temperature": [
            {
                "validator": "range",
                "arguments": ["12.2", "19.8", "0.01"]
            }
        ],
        "resolution": [
            {
                "validator": "enumeration",
                "arguments": ["1080p", "1080i", "720p"]
            }
        ],
        "datatype": [
            {
                "validator": "match_any",
                "validators": [
                    {
                        "validator": "range",
                        "arguments": ["1", "25"]
                    },
                    {
                        "validator": "enumeration",
                        "arguments": ["Collection", "Dataset", "Event"]
                    }
                ]
            }
        ],
```

```
        "topic.vocabulary": [
            {
                "validator": "relationship",
                "arguments": ["animal", "canOnlyHave", "topic.category", "cat", "dog",
"lizard"]
            }
        ]
    }
}
```

1.5. Developing Attribute Types

Create custom attribute types with Attribute Type definition files.

1.5.1. Attribute Type Definition File

To define Attribute Types, the definition file must have an `attributeTypes` key in the root object.

```
{
    "attributeTypes": {...}
}
```

The value of `attributeTypes` must be a map where each key is the attribute type's name and each value is a map that includes the data type and whether the attribute type is stored, indexed, tokenized, or multi-valued.

Attribute Types

```
{
    "attributeTypes": {
        "temperature": {
            "type": "DOUBLE_TYPE",
            "stored": true,
            "indexed": true,
            "tokenized": false,
            "multivalued": false
        }
    }
}
```

The attributes `stored`, `indexed`, `tokenized`, and `multivalued` must be included and must have a boolean value.

2. Required Attribute Definitions

`stored`

If true, the value of the attribute should be stored in the underlying datastore. Some attributes may only be indexed or used in transit and do not need to be persisted.

`indexed`

If true, then the value of the attribute should be included in the datastore's index and therefore be part of query evaluation.

`tokenized`

Only applicable to STRING_TYPE attributes, if true then stopwords and punctuation are stripped prior to storing and/or indexing. If false, only an exact string matches.

`multi-valued`

If true, then the attribute values are Lists of the attribute type rather than single values.

The `type` attribute must also be included and must have one of the allowed values:

3. `type` Attribute Possible Values

- `DATE_TYPE`
- `STRING_TYPE`
- `XML_TYPE`
- `LONG_TYPE`
- `BINARY_TYPE`
- `GEO_TYPE`
- `BOOLEAN_TYPE`
- `DOUBLE_TYPE`
- `FLOAT_TYPE`
- `INTEGER_TYPE`
- `OBJECT_TYPE`
- `SHORT_TYPE`

An example with multiple attributes defined:

Multiple Attributes Defined

```
{
    "attributeTypes": {
```

```
        "resolution": {
            "type": "STRING_TYPE",
            "stored": true,
            "indexed": true,
            "tokenized": false,
            "multivalued": false
        },
        "target-areas": {
            "type": "GEO_TYPE",
            "stored": true,
            "indexed": true,
            "tokenized": false,
            "multivalued": true
        }
    }
}
```

1.6. Developing Default Attribute Types

Create custom default attribute types.

1.6.1. Default Attribute Values

To define default attribute values, the definition file must have a `defaults` key in the root object.

```
{
    "defaults": [...]
}
```

The value of `defaults` is a list of objects where each object contains the keys `attribute`, `value`, and optionally `metacardTypes`.

```
{
    "defaults": [
        {
            "attribute": ...,
            "value": ...,
            "metacardTypes": [...]
        }
    ]
}
```

The value corresponding to the `attribute` key is the name of the attribute to which the default value is applied. The value corresponding to the `value` key is the default value of the attribute.

NOTE	The attribute's default value must be of the same type as the attribute, but it has to be written as a string (enclosed in quotation marks) in the JSON file. Dates must be UTC datetimes in the ISO 8601 format, `yyyy-MM-ddTHH:mm:ssZ`

The `metacardTypes` key is optional. If it is left out, then the default attribute value is applied to every metacard that has that attribute. It can be thought of as a 'global' default value. If the `metacardTypes` key is included, then its value must be a list of strings where each string is the name of a metacard type. In this case, the default attribute value is applied only to metacards that match one of the types given in the list.

NOTE	In the event that an attribute has a 'global' default value as well as a default value for a specific metacard type, the default value for the specific metacard type is applied (the more specific default value wins).

Example:

```
{
    "defaults": [
        {
            "attribute": "title",
            "value": "Default Title"
        },
        {
            "attribute": "description",
            "value": "Default video description",
            "metacardTypes": ["video"]
        },
        {
            "attribute": "expiration",
            "value": "2020-05-06T12:00:00Z",
            "metacardTypes": ["video", "nitf"]
        },
        {
            "attribute": "frame-rate",
            "value": "30"
        }
    ]
}
```

1.7. Developing Attribute Injections

Attribute injections are defined attributes that are injected into all metacard types or into specific metacard types. This capability allows metacard types to be extended with new attributes.

1.7.1. Attribute Injection Definition

To define attribute injections, create a JSON file in the `<DDF_HOME>/etc/definitions` directory. The definition file must have an `inject` key in the root object.

Inject Key

```
{
  "inject": [...]
}
```

The value of `inject` is simply a list of objects where each object contains the key `attribute` and optionally `metacardTypes`.

Inject Values

```
{
  "inject": [
    {
      "attribute": ...,
      "metacardTypes": [...]
    }
  ]
}
```

The value corresponding to the `attribute` key is the name of the attribute to inject.

The `metacardTypes` key is optional. If it is left out, then the attribute is injected into every metacard type. In that case it can be thought of as a 'global' attribute injection. If the `metacardTypes` key is included, then its value must be a list of strings where each string is the name of a metacard type. In this case, the attribute is injected only into metacard types that match one of the types given in the list.

Global and Specific Inject Values

```
{
  "inject": [
    // Global attribute injection, all metacards
    {
      "attribute": "rating"
    },
    // Specific attribute injection, only "video" metacards
    {
      "attribute": "cloud-cover",
      "metacardTypes": "video"
    }
  ]
```

```
}
```

NOTE	Attributes must be registered in the attribute registry (see the `AttributeRegistry` interface) to be injected into metacard types. For example, attributes defined in JSON definition files are placed in the registry, so they can be injected.

Add a second key for `attributeTypes` to register the new types defined previously. For each attribute injections, specify the name and properties for that attribute.

- type: Data type of the possible values for this attribute.
- indexed: Boolean, attribute is indexed.
- stored: Boolean, attribute is stored.
- tokenized: Boolean, attribute is stored.
- multivalued: Boolean, attribute can hold multiple values.

Sample Attribute Injection File

```
{
  "inject": [
    // Global attribute injection, all metacards
    {
      "attribute": "rating"
    },
    // Specific attribute injection, only "video" metacards
    {
      "attribute": "cloud-cover",
      "metacardTypes": "video"
    }
  ],
  "attributeTypes": {
    "rating": {
    "type": "STRING_TYPE",
    "indexed": true,
    "stored": true,
    "tokenized": true,
    "multivalued": true
    },
    "cloud-cover": {
      "type": "STRING_TYPE",
      "indexed": true,
      "stored": true,
      "tokenized": true,
      "multivalued": false
    }
  }
```

```
}
```

1.8. Developing Endpoints

Custom endpoints can be created, if necessary. See Endpoints for descriptions of provided endpoints.

Complete the following procedure to create an endpoint.

1. Create a Java class that implements the endpoint's business logic. Example: Creating a web service that external clients can invoke.
2. Add the 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.*`
4. Retrieve an instance of `CatalogFramework` from the OSGi registry. (Refer to OSGi Basics - Service Registry for examples.)
5. Deploy the packaged service to DDF. (Refer to OSGi Basics - Bundles.)

NOTE	It is recommended to use the maven bundle plugin to create the Endpoint bundle's manifest as opposed to directly editing the manifest file.

TIP	**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.

Table 1. Common Endpoint Business Logic

Methods	Use
`Ingest`	Add, modify, and remove metadata using the ingest-related `CatalogFramework` methods: create, update, and delete.
`Query`	Request metadata using the `query` method.
`Source`	Get available `Source` information.
`Resource`	Retrieve resources referenced in Metacards from Sources.
`Transform`	Convert common Catalog Framework data types to and from other data formats.

1.9. Developing Input Transformers

DDF supports the creation of custom input transformers for use cases not covered by the included implementations.

Creating a custom input Transformer:

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 OSGi Basics section). Export the service to the OSGi Registry and declare service properties.

   Input Transformer 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>
...
```

Table 2. Input Transformer Variable Descriptions / Blueprint Service Properties

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

5. Deploy OSGi Bundle to OSGi runtime.

1.9.1. Create an XML Input Transformer using SaxEventHandlers

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

The following diagrams intend to clarify implementation details:

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

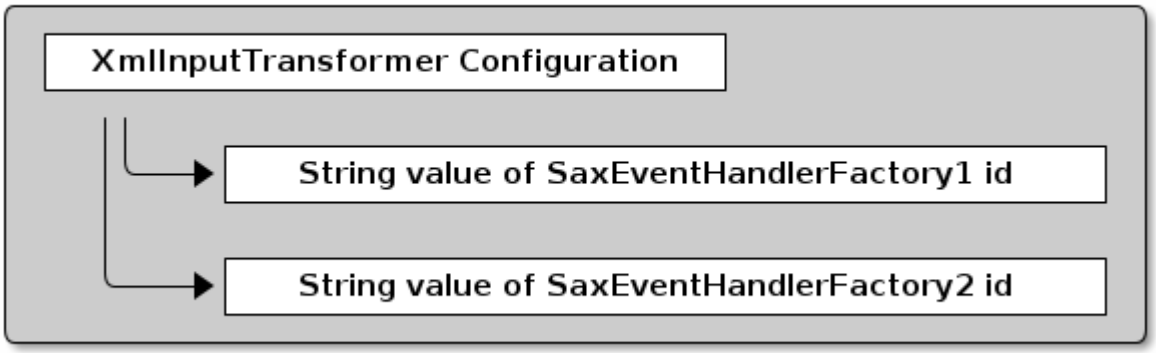

Figure 1. `XMLInputTransformer` *Configuration*

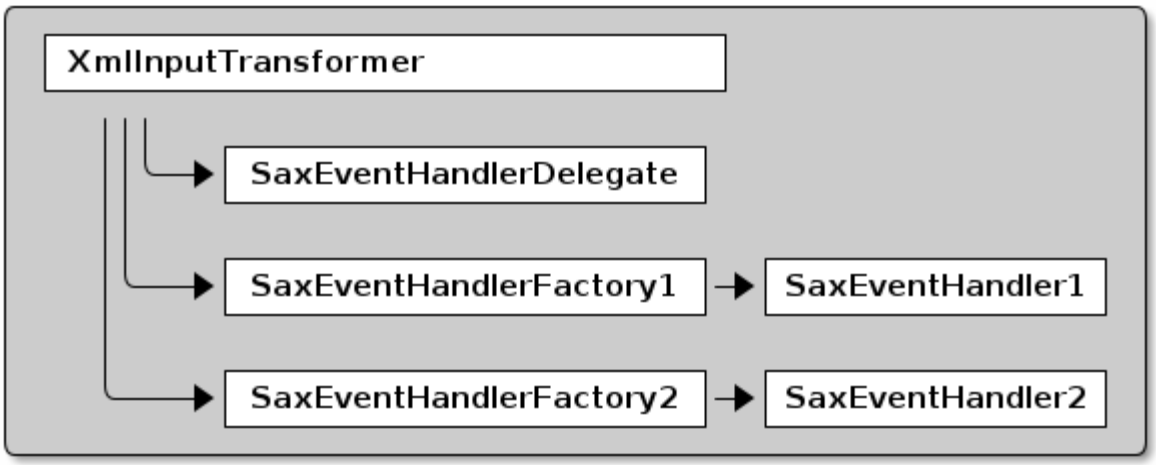

Figure 2. `XMLInputTransformer` `SaxEventHandlerDelegate` *Configuration*

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

NOTE

1. The `XmlInputTransformer` & `SaxEventHandlerDelegate` create and configure themselves based on String matches of the configuration ids with the `SaxEventHandlerFactory` ids, so ensure these match.
2. The `XmlInputTransformer` uses a `DynamicMetacardType`. This is pertinent because a metacards attributes are only stored in the `CatalogProvider` if they are declared on the `MetacardType`. Since the `DynamicMetacardType` is constructed dynamically, attributes are declared by the `SaxEventHandlerFactory` that parses them, as opposed to the `MetacardType`. See `org.codice.ddf.transformer.xml.streaming.impl.XmlSaxEventHandlerFactoryImpl.java` vs `ddf.catalog.data.impl.BasicTypes.java`

1.9.2. Create an Input Transformer Using Apache Camel

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

1.9.2.1. Input Transformer Design Pattern (Camel)

Follow this design pattern for compatibility:

From

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

To

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

Table 3. InputTransformer Message Formats

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

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

InputTransformer Creation Example

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

InputTransformer Creation Details

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

using the DDF custom catalog component that is an InputTransformer registered with an id of application/json;id=geojson.

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

1.9.3. Input Transformer Boot Service Flag

The `org.codice.ddf.platform.bootflag.BootServiceFlag` service with a service property of `id=inputTransformerBootFlag` is used to indicate certain Input Transformers are ready in the system. Adding an Input Transformers ID to a new or existing JSON file under `<DDF_HOME>/etc/transformers` causes the service to wait for an Input Transformer with the given ID.

1.10. Developing Metacard Transformers

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 translates the `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.

1.10.1. Creating a New Metacard Transformer

Existing metacard transformers are written as Java classes, and these steps walk through the steps to create a custom 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`
   a. `transform` must return a `Metacard` or throw an exception. It cannot return null.
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 OSGi Basics section). Export the service to the OSGi registry and declare service properties.

   Metacard Transformer 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.

Table 4. Metacard Transformer Blueprint Service Properties / Variable Descriptions

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

1.11. Developing Query Response Transformers

A Query Response Transformer 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.

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 OSGi Basics). Export the service to the OSGi registry and declare service properties.

5. Deploy OSGi Bundle to OSGi runtime.

Query Response Transformer Blueprint Descriptor Example

```
...
<service ref="SampleResponseTransformer" interface=
"ddf.catalog.transform.QueryResponseTransformer">
    <service-properties>
        <entry key="id" value="[[sampleId]]" />
        <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>
...
```

Table 5. Query Response Transformer Blueprint Service Properties / Variable Descriptions

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

1.12. Developing Sources

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

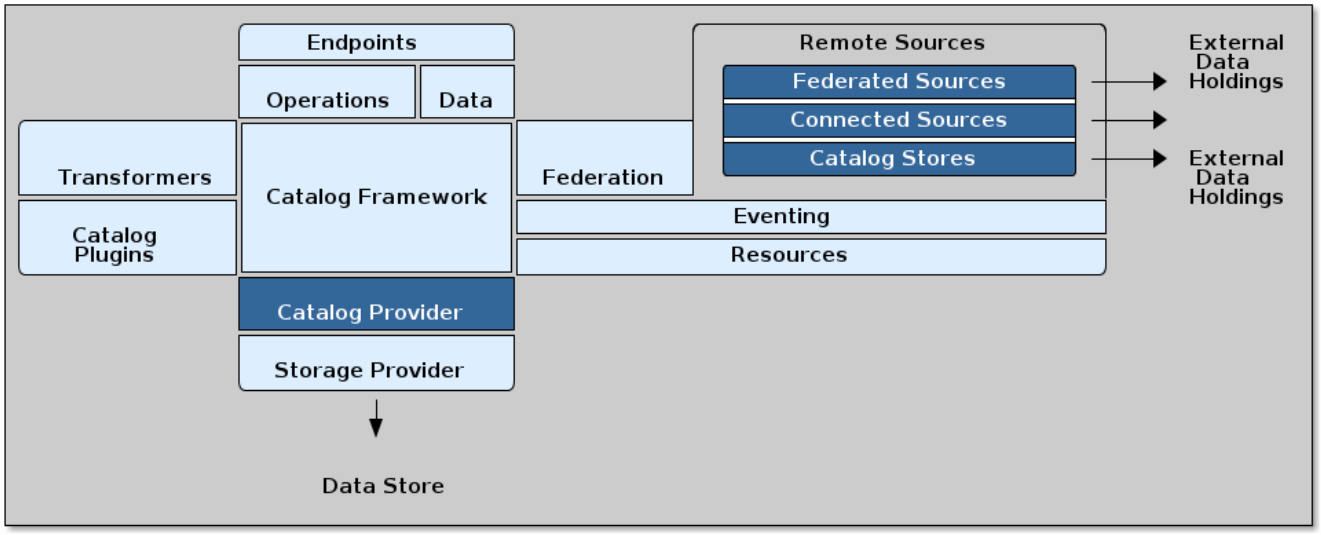

Figure 3. Source Architecture

1.12.1. Implement a Source Interface

There are three types of sources that can be created to perform query operations. All of these sources must also be able to return their availability and the list of content types currently stored in their back-end data stores.

Catalog Provider

`ddf.catalog.source.CatalogProvider` *is used to communicate with back-end storage and allows for Query and Create/Update/Delete operations.*

Federated Source

`ddf.catalog.source.FederatedSource` *is used to communicate with remote systems and only allows query operations.*

Connected Source

`ddf.catalog.source.ConnectedSource` *is 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.*

Catalog Store

`catalog.store.interface` *is used to store data.*

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

1. Create a new class that implements the specified Source interface, the `ConfiguredService` and the required methods.

2. Create an OSGi descriptor file to communicate with the OSGi registry. (Refer to OSGi Services.)
   a. Import DDF packages.
   b. Register source class as service to the OSGi registry.
3. Deploy to DDF.

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

NOTE	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 are called and MUST be properly implemented by a `RemoteSource`.

1.12.1.1. Developing Catalog Providers

Create a custom implementation of a catalog provider.

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

Catalog Provider Blueprint example

```
<service ref="TestCatalogProvider" interface="ddf.catalog.source.CatalogProvider" />
```

See the existing Catalog Provider list for examples of Catalog Providers included in DDF.

1.12.1.2. Developing Federated Sources

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

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

Federated Source Blueprint example

```
<service ref="TestFederatedSource" interface="ddf.catalog.source.FederatedSource" />
```

1.12.1.3. Developing Connected Sources

Create a custom implementation of a connected source.

1. Create a Java class that implements `ConnectedSource` and `ConfiguredService`.
   `public class TestConnectedSource implements ddf.catalog.source.ConnectedSource, ddf.catalog.service.ConfiguredService`
2. Implement the required methods of the `ddf.catalog.source.ConnectedSource` and `ddf.catalog.service.ConfiguredService` interfaces.
3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: ddf.catalog, ddf.catalog.source`
4. Export the service to the OSGi registry.

Connected Source Blueprint example

```
<service ref="TestConnectedSource" interface="ddf.catalog.source.ConnectedSource" />
```

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

1.12.1.4. Exception Handling

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

1.12.1.4.1. Exception Examples

Follow these guidelines for effective exception handling:

- Use a "Site XYZ not found" message rather than the full stack trace with the original site not found exception.

- If 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.
- If 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."

1.12.1.4.2. External Resources for Developing Sources

- Three Rules for Effective Exception Handling.

1.13. Developing Catalog Plugins

Catalog Plugins extend the functionality of the Catalog Framework by performing actions at specified times during a transaction. Plugin interfaces are located in the Catalog Core API. By implementing a plugin interface, actions can be performed at the desired time.

The following types of plugins can be created:

Table 6. Plugin Interfaces

Plugin Type	Plugin Interface	Invocation Order
Pre-Authorization	`ddf.catalog.plugin.PreAuthorizationPlugin`	Before any security rules are applied.
Policy	`ddf.catalog.plugin.PolicyPlugin`	After pre-authorization plugins, but before other catalog plugins to establish the policy for requests/responses.
Access	`ddf.catalog.plugin.AccessPlugin`	Directly after any policy plugins
Pre-Ingest	`ddf.catalog.plugin.PreIngestPlugin`	Before the Create/Update/Delete method is sent to the Catalog Provider.
Post-Ingest	`ddf.catalog.plugin.PostIngestPlugin`	After the Create/Update/Delete method is sent to the Catalog Provider.
Pre-Query	`ddf.catalog.plugin.PreQueryPlugin`	Prior to the Query/Read method being sent to the Source.

Plugin Type	Plugin Interface	Invocation Order
Post-Query	`ddf.catalog.plugin.PostQueryPlugin`	After results have been retrieved from the query but before they are posted to the Endpoint.
Pre-Federated-Query	`ddf.catalog.plugin.PreFederatedQueryPlugin`	Before a federated query is executed.
Post-Federated-Query	`ddf.catalog.plugin.PostFederatedQueryPlugin`	After a federated query has been executed.
Pre-Resource	`ddf.catalog.plugin.PreResourcePlugin`	Prior to a Resource being retrieved.
Post-Resource	`ddf.catalog.plugin.PostResourcePlugin`	After a Resource is retrieved, but before it is sent to the Endpoint.
Pre-Create Storage	`ddf.catalog.content.plugin.PreCreateStoragePlugin`	Experimental Before an item is created in the content repository.
Post-Create Storage	`ddf.catalog.content.plugin.PostCreateStoragePlugin`	Experimental After an item is created in the content repository.
Pre-Update Storage	`ddf.catalog.content.plugin.PreUpdateStoragePlugin`	Experimental Before an item is updated in the content repository.
Post-Update Storage	`ddf.catalog.content.plugin.PostUpdateStoragePlugin`	Experimental After an item is updated in the content repository.
Pre-Subscription	`ddf.catalog.plugin.PreSubscriptionPlugin`	Prior to a Subscription being created or updated.
Pre-Delivery	`ddf.catalog.plugin.PreDeliveryPlugin`	Prior to the delivery of a Metacard when an event is posted.

1.13.1. Implementing Catalog Plugins

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

1. Create a new class that implements the specified plugin interface.
2. Implement the required methods.

3. Create an OSGi descriptor file to communicate with the OSGi registry.
   a. Register the plugin class as a service to OSGi registry.
4. Deploy to DDF.

NOTE	*Plugin Performance Concerns* Plugins should include a check to determine if requests are local or not. It is usually preferable to take no action on non-local requests.

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

1.13.1.1. Catalog Plugin Failure Behavior

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

1.13.1.2. Implementing Pre-Ingest Plugins

Develop a custom Pre-Ingest Plugin.

1. Create a Java class that implements `PreIngestPlugin`.
   `public class SamplePreIngestPlugin implements ddf.catalog.plugin.PreIngestPlugin`
2. Implement the required methods.
   - `public CreateRequest process(CreateRequest input) throws PluginExecutionException, StopProcessingException;`
   - `public UpdateRequest process(UpdateRequest input) throws PluginExecutionException, StopProcessingException;`
   - `public DeleteRequest process(DeleteRequest 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="SamplePreIngestPlugin" interface="ddf.catalog.plugin.PreIngestPlugin" />`

1.13.1.3. Implementing Post-Ingest Plugins

Develop a custom Post-Ingest Plugin.

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

1.13.1.4. Implementing Pre-Query Plugins

Develop a custom Pre-Query Plugin

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

1.13.1.5. Implementing Post-Query Plugins

Develop a custom Post-Query Plugin

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

```
<service ref="SamplePostQueryPlugin" interface="ddf.catalog.plugin.PostQueryPlugin" />
```

1.13.1.6. Implementing Pre-Delivery Plugins

Develop a custom Pre-Delivery Plugin.

1. Create a Java class that implements `PreDeliveryPlugin`.
   `public class SamplePreDeliveryPlugin implements ddf.catalog.plugin.PreDeliveryPlugin`
2. Implement the required methods.
   `public Metacard processCreate(Metacard metacard) throws PluginExecutionException, StopProcessingException; public Update processUpdateMiss(Update update) throws PluginExecutionException, StopProcessingException;`
   - `public Update processUpdateHit(Update update) throws PluginExecutionException, StopProcessingException;`
   - `public Metacard processCreate(Metacard metacard) throws PluginExecutionException, StopProcessingException;`
3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: ddf.catalog,ddf.catalog.plugin,ddf.catalog.operation,ddf.catalog.event`
4. Export the service to the OSGi registry.
   Blueprint descriptor example
   `<service ref="SamplePreDeliveryPlugin" interface="ddf.catalog.plugin.PreDeliveryPlugin" />`

1.13.1.7. Implementing Pre-Subscription Plugins

Develop a custom Pre-Subscription Plugin.

1. Create a Java class that implements `PreSubscriptionPlugin`.
   `public class SamplePreSubscriptionPlugin implements ddf.catalog.plugin.PreSubscriptionPlugin`
2. Implement the required method.
   - `public Subscription process(Subscription input) throws PluginExecutionException, StopProcessingException;`

1.13.1.8. Implementing Pre-Resource Plugins

Develop a custom Pre-Resource Plugin.

1. Create a Java class that implements `PreResourcePlugin`. `public class SamplePreResourcePlugin implements ddf.catalog.plugin.PreResourcePlugin`
2. Implement the required method.
   - `public ResourceRequest process(ResourceRequest input) throws PluginExecutionException, StopProcessingException;`

3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   `Import-Package: ddf.catalog,ddf.catalog.plugin,ddf.catalog.operation`
4. Export the service to the OSGi registry. .Blueprint descriptor example

```
<service ref="SamplePreResourcePlugin" interface="ddf.catalog.plugin.PreResourcePlugin"
/>
```

1.13.1.9. Implementing Post-Resource Plugins

Develop a custom Post-Resource Plugin.

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

Blueprint descriptor example

```
<]]" inter"[[SamplePostResourcePlugin" interface="ddf.catalog.plugin.PostResourcePlugin"
/>
```

1.13.1.10. Implementing Policy Plugins

Develop a custom Policy Plugin.

1. Create a Java class that implements `PolicyPlugin`.
   `public class SamplePolicyPlugin implements ddf.catalog.plugin.PolicyPlugin`
2. Implement the required methods.
   - `PolicyResponse processPreCreate(Metacard input, Map<String, Serializable> properties) throws StopProcessingException;`
   - `PolicyResponse processPreUpdate(Metacard input, Map<String, Serializable> properties) throws StopProcessingException;`
   - `PolicyResponse processPreDelete(String attributeName, List<Serializable> attributeValues, Map<String, Serializable> properties) throws StopProcessingException;`
   - `PolicyResponse processPreQuery(Query query, Map<String, Serializable> properties) throws`

StopProcessingException;

- PolicyResponse processPostQuery(Result input, Map<String, Serializable> properties) throws StopProcessingException;

3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   Import-Package: ddf.catalog,ddf.catalog.plugin,ddf.catalog.operation
4. Export the service to the OSGi registry.
   Blueprint descriptor example
   <]]" inter"[[SamplePolicyPlugin" interface="ddf.catalog.plugin.PolicyPlugin" />

1.13.1.11. Implementing Access Plugins

Develop a custom Access Plugin.

1. Create a Java class that implements AccessPlugin.
   public class SamplePostResourcePlugin implements ddf.catalog.plugin.AccessPlugin
2. Implement the required methods.
   - CreateRequest processPreCreate(CreateRequest input) throws StopProcessingException;
   - UpdateRequest processPreUpdate(UpdateRequest input) throws StopProcessingException;
   - DeleteRequest processPreDelete(DeleteRequest input) throws StopProcessingException;
   - QueryRequest processPreQuery(QueryRequest input) throws StopProcessingException;
   - QueryResponse processPostQuery(QueryResponse input) throws StopProcessingException;
3. Import the DDF interface packages to the bundle manifest (in addition to any other required packages).
   Import-Package: ddf.catalog,ddf.catalog.plugin,ddf.catalog.operation
4. Export the service to the OSGi registry.
   Blueprint descriptor example
   <]]" inter"[[SampleAccessPlugin" interface="ddf.catalog.plugin.AccessPlugin" />

1.14. Developing Token Validators

Token validators are used by the Security Token Service (STS) to validate incoming token requests. The TokenValidator CXF interface must be implemented by all custom token validators. 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 collected from the RST (RequestSecurityToken) message.

1.15. Developing STS Claims Handlers

Develop a custom claims handler to retrieve attributes from an external attribute store.

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

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

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

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

package org.apache.cxf.sts.claims;

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

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

    List<URI> getSupportedClaimTypes();

    ClaimCollection retrieveClaimValues(RequestClaimCollection claims,
ClaimsParameters parameters);
```

```
}
```

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

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

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

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

</blueprint>
```

3. Deploy the bundle.

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

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

STS WS-Trust WSDL Document

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:tns="http://docs.oasis-open.org/ws-sx/ws-trust/200512/"
xmlns:wstrust="http://docs.oasis-open.org/ws-sx/ws-trust/200512/" xmlns:wsdl=
"http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsap10="http://www.w3.org/2006/05/addressing/wsdl" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp=
"http://www.w3.org/ns/ws-policy" xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:wsam=
"http://www.w3.org/2007/05/addressing/metadata" targetNamespace="http://docs.oasis-
open.org/ws-sx/ws-trust/200512/">
    <wsdl:types>
        <xs:schema elementFormDefault="qualified" targetNamespace="http://docs.oasis-
open.org/ws-sx/ws-trust/200512">
            <xs:element name="RequestSecurityToken" type=
"wst:AbstractRequestSecurityTokenType"/>
            <xs:element name="RequestSecurityTokenResponse" type=
"wst:AbstractRequestSecurityTokenType"/>
```

```
            <xs:complexType name="AbstractRequestSecurityTokenType">
                <xs:sequence>
                    <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="Context" type="xs:anyURI" use="optional"/>
                <xs:anyAttribute namespace="##other" processContents="lax"/>
            </xs:complexType>
            <xs:element name="RequestSecurityTokenCollection" type=
"wst:RequestSecurityTokenCollectionType"/>
            <xs:complexType name="RequestSecurityTokenCollectionType">
                <xs:sequence>
                    <xs:element name="RequestSecurityToken" type=
"wst:AbstractRequestSecurityTokenType" minOccurs="2" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
            <xs:element name="RequestSecurityTokenResponseCollection" type=
"wst:RequestSecurityTokenResponseCollectionType"/>
            <xs:complexType name="RequestSecurityTokenResponseCollectionType">
                <xs:sequence>
                    <xs:element ref="wst:RequestSecurityTokenResponse" minOccurs="1"
maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:anyAttribute namespace="##other" processContents="lax"/>
            </xs:complexType>
        </xs:schema>
    </wsdl:types>
    <!-- WS-Trust defines the following GEDs -->
    <wsdl:message name="RequestSecurityTokenMsg">
        <wsdl:part name="request" element="wst:RequestSecurityToken"/>
    </wsdl:message>
    <wsdl:message name="RequestSecurityTokenResponseMsg">
        <wsdl:part name="response" element="wst:RequestSecurityTokenResponse"/>
    </wsdl:message>
    <wsdl:message name="RequestSecurityTokenCollectionMsg">
        <wsdl:part name="requestCollection" element="wst:RequestSecurityTokenCollection
"/>
    </wsdl:message>
    <wsdl:message name="RequestSecurityTokenResponseCollectionMsg">
        <wsdl:part name="responseCollection" element=
"wst:RequestSecurityTokenResponseCollection"/>
    </wsdl:message>
    <!-- This portType an example of a Requestor (or other) endpoint that
         Accepts SOAP-based challenges from a Security Token Service -->
    <wsdl:portType name="WSSecurityRequestor">
        <wsdl:operation name="Challenge">
            <wsdl:input message="tns:RequestSecurityTokenResponseMsg"/>
            <wsdl:output message="tns:RequestSecurityTokenResponseMsg"/>
```

```
        </wsdl:operation>
    </wsdl:portType>
    <!-- This portType is an example of an STS supporting full protocol -->
    <wsdl:portType name="STS">
        <wsdl:operation name="Cancel">
            <wsdl:input wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Cancel" message="tns:RequestSecurityTokenMsg"/>
            <wsdl:output wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RSTR/CancelFinal" message="tns:RequestSecurityTokenResponseMsg"/>
        </wsdl:operation>
        <wsdl:operation name="Issue">
            <wsdl:input wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Issue" message="tns:RequestSecurityTokenMsg"/>
            <wsdl:output wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RSTRC/IssueFinal" message="tns:RequestSecurityTokenResponseCollectionMsg"/>
        </wsdl:operation>
        <wsdl:operation name="Renew">
            <wsdl:input wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Renew" message="tns:RequestSecurityTokenMsg"/>
            <wsdl:output wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RSTR/RenewFinal" message="tns:RequestSecurityTokenResponseMsg"/>
        </wsdl:operation>
        <wsdl:operation name="Validate">
            <wsdl:input wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Validate" message="tns:RequestSecurityTokenMsg"/>
            <wsdl:output wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RSTR/ValidateFinal" message="tns:RequestSecurityTokenResponseMsg"/>
        </wsdl:operation>
        <wsdl:operation name="KeyExchangeToken">
            <wsdl:input wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/KET" message="tns:RequestSecurityTokenMsg"/>
            <wsdl:output wsam:Action="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RSTR/KETFinal" message="tns:RequestSecurityTokenResponseMsg"/>
        </wsdl:operation>
        <wsdl:operation name="RequestCollection">
            <wsdl:input message="tns:RequestSecurityTokenCollectionMsg"/>
            <wsdl:output message="tns:RequestSecurityTokenResponseCollectionMsg"/>
        </wsdl:operation>
    </wsdl:portType>
    <!-- This portType is an example of an endpoint that accepts
         Unsolicited RequestSecurityTokenResponse messages -->
    <wsdl:portType name="SecurityTokenResponseService">
        <wsdl:operation name="RequestSecurityTokenResponse">
            <wsdl:input message="tns:RequestSecurityTokenResponseMsg"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="STS_Binding" type="wstrust:STS">
        <wsp:PolicyReference URI="#STS_policy"/>
```

```
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="Issue">
            <soap:operation soapAction="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Issue"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="Validate">
            <soap:operation soapAction="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Validate"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="Cancel">
            <soap:operation soapAction="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Cancel"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="Renew">
            <soap:operation soapAction="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Renew"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="KeyExchangeToken">
            <soap:operation soapAction="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/KeyExchangeToken"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
```

```
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="RequestCollection">
            <soap:operation soapAction="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/RequestCollection"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsp:Policy wsu:Id="STS_policy">
        <wsp:ExactlyOne>
            <wsp:All>
                <wsap10:UsingAddressing/>
                <wsp:ExactlyOne>
                    <sp:TransportBinding xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702">
                        <wsp:Policy>
                            <sp:TransportToken>
                                <wsp:Policy>
                                    <sp:HttpsToken>
                                        <wsp:Policy/>
                                    </sp:HttpsToken>
                                </wsp:Policy>
                            </sp:TransportToken>
                            <sp:AlgorithmSuite>
                                <wsp:Policy>
                                    <sp:Basic128/>
                                </wsp:Policy>
                            </sp:AlgorithmSuite>
                            <sp:Layout>
                                <wsp:Policy>
                                    <sp:Lax/>
                                </wsp:Policy>
                            </sp:Layout>
                            <sp:IncludeTimestamp/>
                        </wsp:Policy>
                    </sp:TransportBinding>
                </wsp:ExactlyOne>
                <sp:Wss11 xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702">
                    <wsp:Policy>
                        <sp:MustSupportRefKeyIdentifier/>
                        <sp:MustSupportRefIssuerSerial/>
```

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

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

1.15.1. Example Requests and Responses for SAML Assertions

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

1.15.2. BinarySecurityToken (CAS) SAML Security Token Samples

Most endpoints in DDF require the X.509 PublicKey SAML assertion.

BinarySecurityToken (CAS) SAML Security Token Sample Request

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <Action xmlns="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/ws-sx/ws-trust/200512/RST/Issue</Action>
```

```
        <MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:60652909-faca-
4e4a-a4a7-8a5ce243a7cb</MessageID>
        <To xmlns="http://www.w3.org/2005/08/addressing"
>https://server:8993/services/SecurityTokenService</To>
        <ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
            <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
        </ReplyTo>
        <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd" soap:mustUnderstand="1">
            <wsu:Timestamp wsu:Id="TS-1">
                <wsu:Created>2013-04-29T18:35:10.688Z</wsu:Created>
                <wsu:Expires>2013-04-29T18:40:10.688Z</wsu:Expires>
            </wsu:Timestamp>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
        <wst:RequestSecurityToken xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512">
            <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue</wst:RequestType>
            <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
                <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing">
                    <wsa:Address>
https://server:8993/services/SecurityTokenService</wsa:Address>
                </wsa:EndpointReference>
            </wsp:AppliesTo>
            <wst:Claims xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512" Dialect=
"http://schemas.xmlsoap.org/ws/2005/05/identity">
                <ic:ClaimType xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
Optional="true" Uri="
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"/>
                <ic:ClaimType xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
Optional="true" Uri="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
"/>
                <ic:ClaimType xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
Optional="true" Uri="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname"/>
                <ic:ClaimType xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
Optional="true" Uri="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname"/>
                <ic:ClaimType xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
Optional="true" Uri="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"/>
            </wst:Claims>
            <wst:OnBehalfOf>
                <BinarySecurityToken ValueType="#CAS" EncodingType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary" ns1:Id="
CAS" xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd" xmlns:ns1="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
```

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


BinarySecurityToken (CAS) SAML Security Token Sample Response


```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <Action xmlns="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/ws-sx/ws-trust/200512/RSTRC/IssueFinal</Action>
        <MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:7a6fde04-9013-
41ef-b08b-0689ffa9c93e</MessageID>
        <To xmlns="http://www.w3.org/2005/08/addressing"
>http://www.w3.org/2005/08/addressing/anonymous</To>
        <RelatesTo xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:60652909-faca-
4e4a-a4a7-8a5ce243a7cb</RelatesTo>
```

```
        <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd" soap:mustUnderstand="1">
            <wsu:Timestamp wsu:Id="TS-2">
                <wsu:Created>2013-04-29T18:35:11.459Z</wsu:Created>
                <wsu:Expires>2013-04-29T18:40:11.459Z</wsu:Expires>
            </wsu:Timestamp>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
        <RequestSecurityTokenResponseCollection xmlns="http://docs.oasis-open.org/ws-
sx/ws-trust/200512" xmlns:ns2="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" xmlns:ns3="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd" xmlns:ns4="http://www.w3.org/2005/08/addressing"
xmlns:ns5="http://docs.oasis-open.org/ws-sx/ws-trust/200802">
            <RequestSecurityTokenResponse>
                <TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV2.0</TokenType>
                <RequestedSecurityToken>
                    <saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" ID="_BDC44EB8593F47D1B213672605113671" IssueInstant="2013-04-29T18:35:11.370Z"
Version="2.0" xsi:type="saml2:AssertionType">
                        <saml2:Issuer>tokenissuer</saml2:Issuer>
                        <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                            <ds:SignedInfo>
                                <ds:CanonicalizationMethod Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                <ds:SignatureMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                                <ds:Reference URI="#_BDC44EB8593F47D1B213672605113671">
                                    <ds:Transforms>
                                        <ds:Transform Algorithm=
"http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                                        <ds:Transform Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#">
                                            <ec:InclusiveNamespaces xmlns:ec=
"http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="xs"/>
                                        </ds:Transform>
                                    </ds:Transforms>
                                    <ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
                                    <ds:DigestValue>
6wnWbft6Pz5XOF5Q9AG59gcGwLY=</ds:DigestValue>
                                </ds:Reference>
                            </ds:SignedInfo>

<ds:SignatureValue>h+NvkgXGdQtca3/eKebhAKgG38tHp3i2n5uLLy8xXXIg02qyKgEP0FCowp2LiYlsQU9YjK
```

```
fSwCUbH3WR6jhbAv9zj29CE+ePfEny7MeXvgNl3wId+vcHqti/DGGhhgtO2Mbx/tyX1BhHQUwKRlcHajxHeecwmvV
7D85NMdV48tI=</ds:SignatureValue>
                            <ds:KeyInfo>
                                <ds:X509Data>

<ds:X509Certificate>MIIDmjCCAwOgAwIBAgIBBDANBgkqhkiG9w0BAQQFADB1MQswCQYDVQQGEwJVUzEQMA4GA
1UECBMH
QXJpem9uYTERMA8GA1UEBxMIR29vZHllYXIxEDAOBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0V4
YW1wbGUxEDAOBgNVBAsTB0V4YW1wbGUxCzAJBgNVBAMTAkNBMB4XDTEzMDQwOTE4MzcxMVoXDTIz
MDQwNzE4MzcxMVowgaYxCzAJBgNVBAYTAlVTMRAwDgYDVQQIEwdBcml6b25hMREwDwYDVQQHEwhH
b29keWVhcjEQMA4GA1UEChMHRXhhbXBsZTEQMA4GA1UEChMHRXhhbXBsZTEQMA4GA1UECxMHRXhh
bXBsZTEUMBIGA1UEAxMLdG9rZW5pc3N1ZXIxJjAkBgkqhkiG9w0BCQEWF3Rva2VuaXNzdWVyQGV4
YW1wbGUuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDDfktpA8Lrp9rTfRibKdgtxtN9
uB44diiIqq3JOzDGfDhGLu6mjpuHO1hrKItv42hBOhhmH7lS9ipiaQCIpVfgIG63MB7fa5dBrfGF
G69vFrU1Lfi7IvsVVsNrtAEQljOMmw9sxS3SUsRQX+bD8jq7Uj1hpoF7DdqpV8Kb0COOGwIDAQAB
o4IBBjCCAQIwCQYDVR0TBAIwADAsBglghkgBhvhCAQ0EHxYdT3BlblNTTCBHZW5lcmF0ZWQgQ2Vy
dGlmaWNhdGUwHQYDVR0OBBYEFD1mHviop2Tc4HaNu8yPXR6GqWP1MIGnBgNVHSMEgZ8wgZyAFBcn
en6/j05DzaVwORwrteKc7TZOoXmkdzB1MQswCQYDVQQGEwJVUzEQMA4GA1UECBMHQXJpem9uYTER
MA8GA1UEBxMIR29vZHllYXIxEDAOBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0V4YW1wbGUxEDAO
BgNVBAsTB0V4YW1wbGUxCzAJBgNVBAMTAkNBggkAwXk7OcwO7gwwDQYJKoZIhvcNAQEEBQADgYEA
PiTX5kYXwdhmijutSkrObKpRbQkvkkzcyZlO6VrAxRQ+eFeN6NyuyhgYy5K6l/sIWdaGou5iJOQx
2pQYWx1v8Klyl0W22IfEAXYv/epiO89hpdACryuDJpioXI/X8TAwvRwLKL21Dk3k2b+eyCgA0O++
HM0dPfiQLQ99ElWkv/0=</ds:X509Certificate>
                                </ds:X509Data>
                            </ds:KeyInfo>
                        </ds:Signature>
                        <saml2:Subject>
                            <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified" NameQualifier="http://cxf.apache.org/sts">srogers</saml2:NameID>
                            <saml2:SubjectConfirmation Method=
"urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
                                <saml2:SubjectConfirmationData xsi:type=
"saml2:KeyInfoConfirmationDataType">
                                    <ds:KeyInfo xmlns:ds=
"http://www.w3.org/2000/09/xmldsig#">
                                        <ds:X509Data>

<ds:X509Certificate>MIIC5DCCAk2gAwIBAgIJAKj7ROPHjo1yMA0GCSqGSIb3DQEBCwUAMIGKMQswCQYDVQQGE
wJVUzEQ
MA4GA1UECAwHQXJpem9uYTERMA8GA1UEBwwIR29vZHllYXIxGDAWBgNVBAoMD0xvY2toZWVkIE1h
cnRpbjENMAsGA1UECwwESTRDRTEPMA0GA1UEAwwGY2xpZW50MRwwGgYJKoZIhvcNAQkBFg1pNGNl
QGxtY28uY29tMB4XDTEyMDYyMDE5NDMwOVoXDTIyMDYxODE5NDMwOVowgYoxCzAJBgNVBAYTAlVT
MRAwDgYDVQQIDAdBcml6b25hMREwDwYDVQQHDAhHb29keWVhcjEYMBYGA1UECgwPTG9ja2hlZWQg
TWFydGluMQ0wCwYDVQQLDARJNENFMQ8wDQYDVQQDDAZjbGllbnQxHDAaBgkqhkiG9w0BCQEWDWk0
Y2VAbG1jby5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAIpHxCBLYE7xfDLcITS9SsPG
4Q04Z6S32/+TriGsRgpGTj/7GuMG7oJ98m6Ws5cTYl7nyunyHTkZuP7rBzy4esDIHheyx18EgdSJ
vvACgGVCnEmHndkf9bWUlAOfNaxW+vZwljUkRUVdkhPbPdPwOcMdKg/SsLSNjZfsQIjoWd4rAgMB
AAGjUDBOMB0GA1UdDgQWBBQx11VLtYXLvFGpFdHnhlNW9+lxBDAfBgNVHSMEGDAWgBQx11VLtYXL
```

```
vFGpFdHnhlNW9+lxBDAMBgNVHRMEBTADAQH/MA0GCSqGSIb3DQEBCwUAA4GBAHYs2OI0K6yVXzyS
sKcv2fmfw6XCICGTnyA7BOdAjYoqq6wD+33dHJUCFDqye7AWdcivuc7RWJt9jnlfJZKIm2BHcDTR
Hhk6CvjJ14Gf40WQdeMHoX8U8b0diq7Iy5Ravx+zRg7SdiyJUqFYjRh/O5tywXRT1+freI3bwAN0
L6tQ</ds:X509Certificate>
                                        </ds:X509Data>
                                    </ds:KeyInfo>
                                </saml2:SubjectConfirmationData>
                            </saml2:SubjectConfirmation>
                        </saml2:Subject>
                        <saml2:Conditions NotBefore="2013-04-29T18:35:11.407Z"
NotOnOrAfter="2013-04-29T19:05:11.407Z">
                            <saml2:AudienceRestriction>

<saml2:Audience>https://server:8993/services/SecurityTokenService</saml2:Audience>
                            </saml2:AudienceRestriction>
                        </saml2:Conditions>
                        <saml2:AuthnStatement AuthnInstant="2013-04-29T18:35:11.392Z">
                            <saml2:AuthnContext>

<saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified</saml2:Aut
hnContextClassRef>
                            </saml2:AuthnContext>
                        </saml2:AuthnStatement>
                        <saml2:AttributeStatement>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
srogers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string"
>srogers@example.com</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
srogers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">Steve
Rogers</saml2:AttributeValue>
                            </saml2:Attribute>
```

```
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
avengers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
admin</saml2:AttributeValue>
                            </saml2:Attribute>
                        </saml2:AttributeStatement>
                    </saml2:Assertion>
                </RequestedSecurityToken>
                <RequestedAttachedReference>
                    <ns3:SecurityTokenReference xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd" wsse11:TokenType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0">
                        <ns3:KeyIdentifier ValueType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID">
_BDC44EB8593F47D1B213672605113671</ns3:KeyIdentifier>
                    </ns3:SecurityTokenReference>
                </RequestedAttachedReference>
                <RequestedUnattachedReference>
                    <ns3:SecurityTokenReference xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd" wsse11:TokenType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0">
                        <ns3:KeyIdentifier ValueType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID">
_BDC44EB8593F47D1B213672605113671</ns3:KeyIdentifier>
                    </ns3:SecurityTokenReference>
                </RequestedUnattachedReference>
                <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
                    <wsa:EndpointReference xmlns:wsa=
"http://www.w3.org/2005/08/addressing">
                        <wsa:Address>
https://server:8993/services/SecurityTokenService</wsa:Address>
                    </wsa:EndpointReference>
                </wsp:AppliesTo>
                <Lifetime>
                    <ns2:Created>2013-04-29T18:35:11.444Z</ns2:Created>
                    <ns2:Expires>2013-04-29T19:05:11.444Z</ns2:Expires>
                </Lifetime>
            </RequestSecurityTokenResponse>
        </RequestSecurityTokenResponseCollection>
    </soap:Body>
```

```
</soap:Envelope>
```

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

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

1.15.3. UsernameToken Bearer SAML Security Token Sample

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

UsernameToken Bearer SAML Security Token Sample Request

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
        <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd" soap:mustUnderstand="1">
            <wsu:Timestamp wsu:Id="TS-1">
                <wsu:Created>2013-04-29T17:47:37.817Z</wsu:Created>
                <wsu:Expires>2013-04-29T17:57:37.817Z</wsu:Expires>
            </wsu:Timestamp>
            <wsse:UsernameToken wsu:Id="UsernameToken-1">
                <wsse:Username>srogers</wsse:Username>
                <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-username-token-profile-1.0#PasswordText">password1</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
        <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Issue</wsa:Action>
        <wsa:MessageID>uuid:a1bba87b-0f00-46cc-975f-001391658cbe</wsa:MessageID>
        <wsa:To>https://server:8993/services/SecurityTokenService</wsa:To>
    </soap:Header>
    <soap:Body>
        <wst:RequestSecurityToken xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
```

```
trust/200512">
            <wst:SecondaryParameters>
                <t:TokenType xmlns:t="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0</t:TokenType>
                <t:KeyType xmlns:t="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer</t:KeyType>
                <t:Claims xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
xmlns:t="http://docs.oasis-open.org/ws-sx/ws-trust/200512" Dialect=
"http://schemas.xmlsoap.org/ws/2005/05/identity">
                    <!--Add any additional claims you want to grab for the service-->
                    <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/uid"/>
                    <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"/>
                    <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"/>
                    <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"/>
                    <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname"/>
                    <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname"/>
                </t:Claims>
            </wst:SecondaryParameters>
            <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue</wst:RequestType>
            <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
                <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing">
                    <wsa:Address>https://server:8993/services/QueryService</wsa:Address>
                </wsa:EndpointReference>
            </wsp:AppliesTo>
            <wst:Renewing/>
        </wst:RequestSecurityToken>
    </soap:Body>
</soap:Envelope>
```

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

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

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

The `AttributeStatement` block contains all the Claims requested.

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

UsernameToken Bearer SAML Security Token Sample Response

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

```
                                            <ec:InclusiveNamespaces xmlns:ec=
"http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="xs"/>
                                        </ds:Transform>
                                    </ds:Transforms>
                                    <ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
                                    <ds:DigestValue>
ReOqEbGZlyplW5kqiynXOjPnVEA=</ds:DigestValue>
                                </ds:Reference>
                            </ds:SignedInfo>

<ds:SignatureValue>X5Kzd54PrKIlGVV2XxzCmWFRzHRoybF7hU6zxbEhSLMR0AWS9R7Me3epq91XqeOwvIDDbw
mE/oJNC7vI0fIw/rqXkx4aZsY5a5nbAs7f+aXF9TGdk82x2eNhNGYpViq0YZJfsJ5WSyMtG8w5nRekmDMy9oTLsHG
+Y/OhJDEwq58=</ds:SignatureValue>
                            <ds:KeyInfo>
                                <ds:X509Data>

<ds:X509Certificate>MIIDmjCCAwOgAwIBAgIBBDANBgkqhkiG9w0BAQQFADB1MQswCQYDVQQGEwJVUzEQMA4GA
1UECBMH
QXJpem9uYTERMA8GA1UEBxMIR29vZHllYXIxEDAOBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0V4
YW1wbGUxEDAOBgNVBAsTB0V4YW1wbGUxCzAJBgNVBAMTAkNBMB4XDTEzMDQwOTE4MzcxMVoXDTIz
MDQwNzE4MzcxMVowgaYxCzAJBgNVBAYTAlVTMRAwDgYDVQQIEwdBcml6b25hMREwDwYDVQQHEwhH
b29keWVhcjEQMA4GA1UEChMHRXhhbXBsZTEQMA4GA1UEChMHRXhhbXBsZTEQMA4GA1UECxMHRXhh
bXBsZTEUMBIGA1UEAxMLdG9rZW5pc3N1ZXIxJjAkBgkqhkiG9w0BCQEWF3Rva2VuaXNzdWVyQGV4
YW1wbGUuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDDfktpA8Lrp9rTfRibKdgtxtN9
uB44diiIqq3JOzDGfDhGLu6mjpuHO1hrKItv42hBOhhmH7lS9ipiaQCIpVfgIG63MB7fa5dBrfGF
G69vFrU1Lfi7IvsVVsNrtAEQljOMmw9sxS3SUsRQX+bD8jq7Uj1hpoF7DdqpV8Kb0COOGwIDAQAB
o4IBBjCCAQIwCQYDVR0TBAIwADAsBglghkgBhvhCAQ0EHxYdT3BlblNTTCBHZW5lcmF0ZWQgQ2Vy
dGlmaWNhdGUwHQYDVR0OBBYEFD1mHviop2Tc4HaNu8yPXR6GqWP1MIGnBgNVHSMEgZ8wgZyAFBcn
en6/j05DzaVwORwrteKc7TZOoXmkdzB1MQswCQYDVQQGEwJVUzEQMA4GA1UECBMHQXJpem9uYTER
MA8GA1UEBxMIR29vZHllYXIxEDAOBgNVBAoTB0V4YW1wbGUxEDAOBgNVBAoTB0V4YW1wbGUxEDAO
BgNVBAsTB0V4YW1wbGUxCzAJBgNVBAMTAkNBggkAwXk7OcwO7gwwDQYJKoZIhvcNAQEEBQADgYEA
PiTX5kYXwdhmijutSkrObKpRbQkvkkzcyZlO6VrAxRQ+eFeN6NyuyhgYy5K6l/sIWdaGou5iJOQx
2pQYWx1v8Klyl0W22IfEAXYv/epiO89hpdACryuDJpioXI/X8TAwvRwLKL21Dk3k2b+eyCgA0O++
HM0dPfiQLQ99ElWkv/0=</ds:X509Certificate>
                                </ds:X509Data>
                            </ds:KeyInfo>
                        </ds:Signature>
                        <saml2:Subject>
                            <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified" NameQualifier="http://cxf.apache.org/sts">srogers</saml2:NameID>
                            <saml2:SubjectConfirmation Method=
"urn:oasis:names:tc:SAML:2.0:cm:bearer"/>
                        </saml2:Subject>
                        <saml2:Conditions NotBefore="2013-04-29T17:49:12.614Z"
NotOnOrAfter="2013-04-29T18:19:12.614Z">
                            <saml2:AudienceRestriction>
                                <saml2:Audience>
```

```
https://server:8993/services/QueryService</saml2:Audience>
                            </saml2:AudienceRestriction>
                        </saml2:Conditions>
                        <saml2:AuthnStatement AuthnInstant="2013-04-29T17:49:12.613Z">
                            <saml2:AuthnContext>

<saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified</saml2:Aut
hnContextClassRef>
                            </saml2:AuthnContext>
                        </saml2:AuthnStatement>
                        <saml2:AttributeStatement>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
srogers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string"
>srogers@example.com</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
srogers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">Steve
Rogers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
avengers</saml2:AttributeValue>
                            </saml2:Attribute>
                            <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                                <saml2:AttributeValue xsi:type="xs:string">
admin</saml2:AttributeValue>
                            </saml2:Attribute>
                        </saml2:AttributeStatement>
```

```
                    </saml2:Assertion>
                </RequestedSecurityToken>
                <RequestedAttachedReference>
                    <ns3:SecurityTokenReference xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd" wsse11:TokenType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0">
                        <ns3:KeyIdentifier ValueType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID">
_7437C1A55F19AFF22113672577526132</ns3:KeyIdentifier>
                    </ns3:SecurityTokenReference>
                </RequestedAttachedReference>
                <RequestedUnattachedReference>
                    <ns3:SecurityTokenReference xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd" wsse11:TokenType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0">
                        <ns3:KeyIdentifier ValueType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID">
_7437C1A55F19AFF22113672577526132</ns3:KeyIdentifier>
                    </ns3:SecurityTokenReference>
                </RequestedUnattachedReference>
                <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
                    <wsa:EndpointReference xmlns:wsa=
"http://www.w3.org/2005/08/addressing">
                        <wsa:Address>
https://server:8993/services/QueryService</wsa: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>
```


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

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

1.15.4. X.509 PublicKey SAML Security Token Sample

X.509 PublicKey SAML Security Token Request

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

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

X.509 PublicKey SAML Security Token Sample Request

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

         <!-- OPTIONAL: Only required if the endpoint that the SAML assertion will be
sent to requires claims. -->
         <wsse:UsernameToken wsu:Id="UsernameToken-16">
            <wsse:Username>pparker</wsse:Username>
            <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordText">password1</wsse:Password>
            <wsse:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-soap-message-security-1.0#Base64Binary">LCTD+5Y7hlWIP6SpsEg9XA==</wsse:Nonce>
```

```
            <wsu:Created>2014-02-19T17:30:37.355Z</wsu:Created>
         </wsse:UsernameToken>
      </wsse:Security>
   </soapenv:Header>
   <soapenv:Body>
      <wst:RequestSecurityToken xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512">
         <wst:TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV2.0</wst:TokenType>
         <wst:KeyType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/PublicKey</wst:KeyType>

         <!-- OPTIONAL: Only required if the endpoint that the SAML assertion will be
sent to requires claims. -->
         <wst:Claims Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity" xmlns:ic=
"http://schemas.xmlsoap.org/ws/2005/05/identity">
            <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"/>
            <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"/>
            <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"/>
            <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname"/>
            <ic:ClaimType Optional="true" Uri=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname"/>
         </wst:Claims>
         <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue</wst:RequestType>
            <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
            <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing">
            <wsa:Address>https://server:8993/services/QueryService</wsa:Address>
            </wsa:EndpointReference>
         </wsp:AppliesTo>
         <wst:UseKey>
            <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
               <ds:X509Data>

<ds:X509Certificate>MIIFGDCCBACgAwIBAgICJe0wDQYJKoZIhvcNAQEFBQAwXDELMAkGA1UEBhMCVVMxGDAWB
gNVBAoT
D1UuUy4gR292ZXJubWVudDEMMAoGA1UECxMDRG9EMQwwCgYDVQQLEwNQS0kxFzAVBgNVBAMTDkRP
RCBKSVRDIENBLTI3MB4XDTEzMDUwNzAwMjU0OVoXDTE2MDUwNzAwMjU0OVowaTELMAkGA1UEBhMC
VVMxGDAWBgNVBAoTD1UuUy4gR292ZXJubWVudDEMMAoGA1UECxMDRG9EMQwwCgYDVQQLEwNQS0kx
EzARBgNVBAsTCkNPTlRSQUNUT1IxDzANBgNVBAMTBmNsaWVudDCCASIwDQYJKoZIhvcNAQEBBQAD
ggEPADCCAQoCggEBAOq6L1/jjZ5cyhjhHEbOHr5WQpboKACYbrsn8lg85LGNoAfcwImr9KBmOxGb
ZCxHYIhkW7pJ+kppyH8DbbbDMviIvvdkvrAIU0l8OBRn2wReCBGQ01Imdc3+WzFF2svW75d6wii2ZVd
eMvUO15p/pAD/sdIfXmAfyu8+tqtiO8KVZGkTnlg3AMzfeSrkci5UHMVWj0qUSuzLk9SAg/9STgb
Kf2xBpHUYecWFSB+dTpdZN2pC85tj9xIoWGh5dFWG1fPcYRgzGPxsybiGOylbJ7rHDJuL7IIIyx5
```

```
EnkCuxmQwoQ6XQAhiWRGyPlY08w1LZixI2v+Cv/ZjUfIHv49I9P4Mt8CAwEAAaOCAdUwggHRMB8G
A1UdIwQYMBaAFCMUNCBNXy43NZLBBlnDjDplNZJoMB0GA1UdDgQWBBRPGiX6zZzKTqQSx/tjg6hx
9opDoTAOBgNVHQ8BAf8EBAMCBaAwgdoGA1UdHwSB0jCBzzA2oDSgMoYwaHR0cDovL2NybC5nZHMu
bml0LmRpc2EubWlsL2NybC9ET0RKSVRDQ0FfMjcuY3JsMIGUoIGRoIGOhoGLbGRhcDovL2NybC5n
ZHMubml0LmRpc2EubWlsL2NuJTNkRE9EJTIwSklUQyUyMENBLTI3JTJjb3UlM2RQS0klMmNvdSUz
ZERvRCUyY28lM2RVLlMuJTIwR292ZXJubWVudCUyY2MlM2RVUz9jZXJ0aWZpY2F0ZXJldm9jYXRp
b25saXN0O2JpbmFyeTAjBgNVHSAEHDAaMAsGCWCGSAFlAgELBTALBglghkgBZQIBCxIwfQYIKwYB
BQUHAQEEcTBvMD0GCCsGAQUFBzAChjFodHRwOi8vY3JsLmdkcy5uaXQuZGlzYS5taWwvc2lnbi9E
T0RKSVRDQ0FfMjcuY2VyMC4GCCsGAQUFBzABhiJodHRwOi8vb2NzcC5uc24wLnJjdnMubml0LmRp
c2EubWlsMA0GCSqGSIb3DQEBBQUAA4IBAQCGUJPGh4iGCbr2xCMqCq04SFQ+iaLmTIFAxZPFvup1
4E9Ir6CSDalpF9eBx9fS+Z2xuesKyM/g3YqWU1LtfWGRRIxzEujaC4YpwHuffkx9QqkwSkXXIsim
EhmzSgzxnT4Q9X8WwalqVYOfNZ6sSLZ8qPPFrLHkkw/zIFRzo62wXLu0tfcpOr+iaJBhyDRinIHr
hwtE3xo6qQRRWlO3/clC4RnTev1crFVJQVBF3yfpRu8udJ2SOGdqU0vjUSu1h7aMkYJMHIu08Whj
8KASjJBFeHPirMV1oddJ5ydZCQ+Jmnpbwq+XsCxg1LjC4dmbjKVr9s4QK+/JLNjxD8IkJiZE</ds:X509Certific
ate>
                </ds:X509Data>
            </ds:KeyInfo>
         </wst:UseKey>
      </wst:RequestSecurityToken>
   </soapenv:Body>
</soapenv:Envelope>
```


X.509 PublicKey SAML Security Token Response

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

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

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

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

X.509 PublicKey SAML Security Token Sample Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header>
      <Action xmlns="http://www.w3.org/2005/08/addressing">http://docs.oasis-open.org/ws-
sx/ws-trust/200512/RSTRC/IssueFinal</Action>
      <MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:b46c35ad-3120-
4233-ae07-b9e10c7911f3</MessageID>
      <To xmlns="http://www.w3.org/2005/08/addressing"
>http://www.w3.org/2005/08/addressing/anonymous</To>
      <RelatesTo xmlns="http://www.w3.org/2005/08/addressing">uuid:527243af-94bd-4b5c-
a1d8-024fd7e694c5</RelatesTo>
```

```
      <wsse:Security soap:mustUnderstand="1" xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu=
"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
         <wsu:Timestamp wsu:Id="TS-90DBA0754E55B4FE7013928310431357">
            <wsu:Created>2014-02-19T17:30:43.135Z</wsu:Created>
            <wsu:Expires>2014-02-19T17:35:43.135Z</wsu:Expires>
         </wsu:Timestamp>
      </wsse:Security>
   </soap:Header>
   <soap:Body>
      <ns2:RequestSecurityTokenResponseCollection xmlns="http://docs.oasis-open.org/ws-
sx/ws-trust/200802" xmlns:ns2="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
xmlns:ns3="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd" xmlns:ns4="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" xmlns:ns5="http://www.w3.org/2005/08/addressing">
         <ns2:RequestSecurityTokenResponse>
            <ns2:TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV2.0</ns2:TokenType>
            <ns2:RequestedSecurityToken>
               <saml2:Assertion ID="_90DBA0754E55B4FE7013928310431176" IssueInstant=
"2014-02-19T17:30:43.117Z" Version="2.0" xsi:type="saml2:AssertionType" xmlns:saml2=
"urn:oasis:names:tc:SAML:2.0:assertion" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                  <saml2:Issuer>tokenissuer</saml2:Issuer>
                  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                     <ds:SignedInfo>
                        <ds:CanonicalizationMethod Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#"/>
                        <ds:SignatureMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                        <ds:Reference URI="#_90DBA0754E55B4FE7013928310431176">
                           <ds:Transforms>
                              <ds:Transform Algorithm=
"http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                              <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#">
                                 <ec:InclusiveNamespaces PrefixList="xs" xmlns:ec=
"http://www.w3.org/2001/10/xml-exc-c14n#"/>
                              </ds:Transform>
                           </ds:Transforms>
                           <ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
                           <ds:DigestValue>/bEGqsRGHVJbx298WPmGd8I53zs=</ds:DigestValue>
                        </ds:Reference>
                     </ds:SignedInfo>
                     <ds:SignatureValue>
mYR7w1/dnuh8Z7t9xjCb4XkYQLshj+UuYlGOuTwDYsUPcS2qI0nAgMD1VsDP7y1fDJxeqsq7HYhFKsnqRfebMM4WL
H1D/lJ4rD4UO+i9l3tuiHml7SN24WM1/bOqfDUCoDqmwG8afUJ3r4vmTNPxfwfOss8BZ/8ODgZzm08ndlkxDfvcN7
```

OrExbV/3/45JwF/MMPZoqvi2MJGfX56E9fErJNuzezpWnRqPOlWPxyffKMAlVaB9zF6gvVnUqcW2k/Z8X9lN7O5jo
uBI281ZnIfsIPuBJERFtYNVDHsIXM1pJnrY6FlKIaOsi55LQu3Ruir/n82pU7BT5aWtxwrn7akBg==
</ds:SignatureValue>
<ds:KeyInfo>
<ds:X509Data>

<ds:X509Certificate>MIIFHTCCBAWgAwIBAgICJe8wDQYJKoZIhvcNAQEFBQAwXDELMAkGA1UEBhMCVVMxGDAWB
gNVBAoT
D1UuUy4gR292ZXJubWVudDEMMAoGA1UECxMDRG9EMQwwCgYDVQQLEwNQS0kxFzAVBgNVBAMTDkRP
RCBKSVRDIENBLTI3MB4XDTEzMDUwNzAwMjYzN1oXDTE2MDUwNzAwMjYzN1owbjELMAkGA1UEBhMC
VVMxGDAWBgNVBAoTD1UuUy4gR292ZXJubWVudDEMMAoGA1UECxMDRG9EMQwwCgYDVQQLEwNQS0kx
EzARBgNVBAsTCkNPTlRSQUNUT1IxFDASBgNVBAMTC3Rva2VuaXNzdWVyMIIBIjANBgkqhkiG9w0B
AQEFAAOCAQ8AMIIBCgKCAQEAx01/U4M1wG+wL1JxX2RL1glj101FkJXMk3KFt3zD//N8x/Dcwwvs
ngCQjXrV6YhbB2V7scHwnThPv3RSwYYiO62z+g6ptfBbKGGBLSZOzLe3fyJR4RxblFKsELFgPHfX
vgUHS/keG5uSRk9S/Okqps/yxKB7+ZlxeFxsIz5QywXvBpMiXtc2zF+M7BsbSIdSx5LcPcDFBwjF
c66rE3/y/25VMht9EZX1QoKr7f8rWD4xgd5J6DYMFWEcmiCz4BDJH9sfTw+n1P+CYgrhwslWGqxt
cDME9t6SWR3GLT4Sdtr8ziIM5uUteEhPIV3rVC3/u23JbYEeS8mpnp0bxt5eHQIDAQABo4IB1TCC
AdEwHwYDVR0jBBgwFoAUIxQ0IE1fLjc1ksEGWcOMOmU1kmgwHQYDVR0OBBYEFGBjdkdey+bMHMhC
Z7gwiQ/mJf5VMA4GA1UdDwEB/wQEAwIFoDCB2gYDVR0fBIHSMIHPMDagNKAyhjBodHRwOi8vY3Js
Lmdkcy5uaXQuZGlzYS5taWwvY3JsL0RPREpJVENDQV8yNy5jcmwwgZSggZGggY6GgYtsZGFwOi8v
Y3JsLmdkcy5uaXQuZGlzYS5taWwvY24lM2RET0QlMjBKSVRDJTIwQ0EtMjclMmNvdSUzZFBLSSUy
Y291JTNkRG9EJTJjbyUzZFUuUy4lMjBHb3Zlcm5tZW50JTJjYyUzZFVTP2NlcnRpZmljYXRlcmV2
b2NhdGlvbmxpc3Q7YmluYXJ5MCMGA1UdIAQcMBowCwYJYIZIAWUCAQsFMAsGCWCGSAFlAgELEjB9
BggrBgEFBQcBAQRxMG8wPQYIKwYBBQUHMAKGMWh0dHA6Ly9jcmwuZ2RzLm5pdC5kaXNhLm1pbC9z
aWduL0RPREpJVENDQV8yNy5jZXIwLgYIKwYBBQUHMAGGImh0dHA6Ly9vY3NwLm5zbjAucmN2cy5u
aXQuZGlzYS5taWwwDQYJKoZIhvcNAQEFBQADggEBAIHZQTINU3bMpJ/PkwTYLWPmwCqAYgEUzSYx
bNcVY5MWD8b4XCdw5nM3GnFlOqr4IrHeyyOzsEbIebTe3bv0l1pHx0Uyj059nAhx/AP8DjVtuRU1
/Mp4b6uJ/4yaoMjIGceqBzHqhHIJinG0Y2azua7eM9hVbWZsa912ihbiupCq22mYuHFP7NUNzBvV
j03YUcsy/sES5sRx9Rops/CBN+LUUYOdJOxYWxo8oAbtF8ABE5ATLAwqz4ttsToKPUYh1sxdx5Ef
APeZ+wYDmMu4OfLckwnCKZgkEtJOxXpdIJHY+VmyZtQSB0LkR5toeH/ANV4259Ia5ZT8h2/vIJBg
6B4=</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
</ds:Signature>
<saml2:Subject>
<saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified" NameQualifier="http://cxf.apache.org/sts">pparker</saml2:NameID>
<saml2:SubjectConfirmation Method=
"urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
<saml2:SubjectConfirmationData xsi:type=
"saml2:KeyInfoConfirmationDataType">
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:X509Data>

<ds:X509Certificate>MIIFGDCCBACgAwIBAgICJe0wDQYJKoZIhvcNAQEFBQAwXDELMAkGA1UEBhMCVVMxGDAWB
gNVBAoT
D1UuUy4gR292ZXJubWVudDEMMAoGA1UECxMDRG9EMQwwCgYDVQQLEwNQS0kxFzAVBgNVBAMTDkRP
RCBKSVRDIENBLTI3MB4XDTEzMDUwNzAwMjU0OVoXDTE2MDUwNzAwMjU0OVowaTELMAkGA1UEBhMC

```
VVMxGDAWBgNVBAoTD1UuUy4gR292ZXJubWVudDEMMAoGA1UECxMDRG9EMQwwCgYDVQQLEwNQS0kx
EzARBgNVBAsTCkNPTlRSQUNUT1IxDzANBgNVBAMTBmNsaWVudDCCASIwDQYJKoZIhvcNAQEBBQAD
ggEPADCCAQoCggEBAOq6L1/jjZ5cyhjhHEbOHr5WQpboKACYbrsn8lg85LGNoAfcwImr9KBmOxGb
ZCxHYIhkW7pJ+kppyH8bbbviIvvdkvrAIU0l8OBRn2wReCBGQ01Imdc3+WzFF2svW75d6wii2ZVd
eMvUO15p/pAD/sdIfXmAfyu8+tqtiO8KVZGkTnlg3AMzfeSrkci5UHMVWj0qUSuzLk9SAg/9STgb
Kf2xBpHUYecWFSB+dTpdZN2pC85tj9xIoWGh5dFWG1fPcYRgzGPxsybiGOylbJ7rHDJuL7IIIyx5
EnkCuxmQwoQ6XQAhiWRGyPlY08w1LZixI2v+Cv/ZjUfIHv49I9P4Mt8CAwEAAaOCAdUwggHRMB8G
A1UdIwQYMBaAFCMUNCBNXy43NZLBBlnDjDplNZJoMB0GA1UdDgQWBBRPGiX6zZzKTqQSx/tjg6hx
9opDoTAOBgNVHQ8BAf8EBAMCBaAwgdoGA1UdHwSB0jCBzzA2oDSgMoYwaHR0cDovL2NybC5nZHMu
bml0LmRpc2EubWlsL2NybC9ET0RKSVRDQ0FfMjcuY3JsMIGUoIGRoIGOhoGLbGRhcDovL2NybC5n
ZHMubml0LmRpc2EubWlsL2NuJTNkRE9EJTIwSklUQyUyMENBLTI3JTJjb3UlM2RQS0klMmNvdSUz
ZERvRCUyY28lM2RVLlMuJTIwR292ZXJubWVudCUyY2MlM2RVUz9jZXJ0aWZpY2F0ZXJldm9jYXRp
b25saXN0O2JpbmFyeTAjBgNVHSAEHDAaMAsGCWCGSAFlAgELBTALBglghkgBZQIBCxIwfQYIKwYB
BQUHAQEEcTBvMD0GCCsGAQUFBzAChjFodHRwOi8vY3JsLmdkcy5uaXQuZGlzYS5taWwvc2lnbi9E
T0RKSVRDQ0FfMjcuY2VyMC4GCCsGAQUFBzABhiJodHRwOi8vb2NzcC5uc24wLnJjdnMubml0LmRp
c2EubWlsMA0GCSqGSIb3DQEBBQUAA4IBAQCGUJPGh4iGCbr2xCMqCq04SFQ+iaLmTIFAxZPFvup1
4E9Ir6CSDalpF9eBx9fS+Z2xuesKyM/g3YqWU1LtfWGRRIxzEujaC4YpwHuffkx9QqkwSkXXIsim
EhmzSgzxnT4Q9X8WwalqVYOfNZ6sSLZ8qPPFrLHkkw/zIFRzo62wXLu0tfcpOr+iaJBhyDRinIHr
hwtE3xo6qQRRWlO3/clC4RnTev1crFVJQVBF3yfpRu8udJ2SOGdqU0vjUSu1h7aMkYJMHIu08Whj
8KASjJBFeHPirMV1oddJ5ydZCQ+Jmnpbwq+XsCxg1LjC4dmbjKVr9s4QK+/JLNjxD8IkJiZE</ds:X509Certific
ate>
                            </ds:X509Data>
                         </ds:KeyInfo>
                      </saml2:SubjectConfirmationData>
                   </saml2:SubjectConfirmation>
                </saml2:Subject>
                <saml2:Conditions NotBefore="2014-02-19T17:30:43.119Z" NotOnOrAfter=
"2014-02-19T18:00:43.119Z"/>
                <saml2:AuthnStatement AuthnInstant="2014-02-19T17:30:43.117Z">
                   <saml2:AuthnContext>

<saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified</saml2:Aut
hnContextClassRef>
                   </saml2:AuthnContext>
                </saml2:AuthnStatement>

                <!-- This block will only be included if Claims were requested in the
RST. -->
                <saml2:AttributeStatement>
                   <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                      <saml2:AttributeValue xsi:type="xs:string">
pparker</saml2:AttributeValue>
                   </saml2:Attribute>
                   <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
```

```
                        <saml2:AttributeValue xsi:type="xs:string">
pparker@example.com</saml2:AttributeValue>
                     </saml2:Attribute>
                     <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">
pparker</saml2:AttributeValue>
                     </saml2:Attribute>
                     <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">Peter
Parker</saml2:AttributeValue>
                     </saml2:Attribute>
                     <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">
users</saml2:AttributeValue>
                     </saml2:Attribute>
                     <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">
users</saml2:AttributeValue>
                     </saml2:Attribute>
                     <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">
avengers</saml2:AttributeValue>
                     </saml2:Attribute>
                     <saml2:Attribute Name=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
                        <saml2:AttributeValue xsi:type="xs:string">
admin</saml2:AttributeValue>
                     </saml2:Attribute>
                  </saml2:AttributeStatement>
               </saml2:Assertion>
            </ns2:RequestedSecurityToken>
            <ns2:RequestedAttachedReference>
               <ns4:SecurityTokenReference wsse11:TokenType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0" xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd">
                  <ns4:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-wss-
saml-token-profile-1.1#SAMLID">_90DBA0754E55B4FE7013928310431176</ns4:KeyIdentifier>
```

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

1.16. Developing Registry Clients

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.

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`.

1.17. Developing Resource Readers

A Resource Reader is a class that retrieves a resource from a native/external source and returns it to DDF. A simple example is that of a File `ResourceReader`. It takes a file from the local file system and passes it back to DDF. New implementations can be created in order to support obtaining Resources from various Resource data stores.

1.17.1. Creating a New `ResourceReader`

Complete the following procedure to create a `ResourceReader`.

1. Create a Java class that implements the `DDF.catalog.resource.ResourceReader` interface.
2. Deploy the OSGi bundled packaged service to the DDF run-time.

1.17.1.1. Implementing the `ResourceReader` Interface

```
public class TestResourceReader implements DDF.catalog.resource.ResourceReader
```

`ResourceReader` has a couple of key methods where most of the work is performed.

NOTE	**URI** It is recommended to become familiar with the Java API URI class in order to properly build a `ResourceReader`. Furthermore, a URI should be used according to its specification.

1.17.1.2. 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 needs 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` are 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.

The `URLResourceReader` is an example `ResourceReader` that reads a file from a URI.

NOTE	The `Map<String, Serializable> arguments` parameter is passed in to support any options or additional information associated with retrieving the resource.

1.17.1.3. Implement `retrieveResource()`

1. Define supported schemes (such as file, http, etc.).
2. Check if the incoming URI matches a supported scheme. If it does not, throw `ResourceNotSupportedException`.

Example:

```
if ( !uri.getScheme().equals("http") )
 {
   throw new ResourceNotSupportedException("Unsupported scheme received, was expecting
http")
 }
```

1. Implement the business logic.
2. For example, the `URLResourceReader` obtains the resource through a connection:

```
URL url = uri.toURL();
URLConnection conn = url.openConnection();
String mimeType = conn.getContentType();
if ( mimeType == null ) {
    mimeType = URLConnection.guessContentTypeFromName( url.getFile() );
}
InputStream is = conn.getInputStream();
```

NOTE	The `Resource` needs to be accessible from the DDF installation (see the rootResourceDirectories property of the `URLResourceReader`). This includes being able to find a file locally or reach out to a remote URI. This may require Internet access, and DDF may need to be configured to use a proxy (`http.proxyHost` and `http.proxyPort` can be added to the system properties on the command line script).

1. Return `Resource` in `ResourceResponse`.

For example:

```
return ResourceResponseImpl( new ResourceImpl( new BufferedInputStream( is ), new
MimeType( mimeType ), url.getFile() ) );
```

If the Resource cannot be found, throw a `ResourceNotFoundException`.

1.17.1.4. `getSupportedSchemes`

```
public Set<String> getSupportedSchemes();
```

This method lets the `ResourceReader` inform the CatalogFramework about the type of URI scheme that it accepts and should be passed. For single-use ResourceReaders (like a URLResourceReader), there may be only one scheme that it can accept while others may understand more than one. A ResourceReader must, at minimum, accept one qualifier. As mentioned before, this method is used by the

`CatalogFramework` to determine which `ResourceReader` to invoke.

NOTE	**`ResourceReader` extends `Describable`** Additionally, there are other methods that are used to uniquely describe a `ResourceReader`. The `describe` methods are straight-forward and can be implemented with guidance from the Javadoc.

1.17.1.5. 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" />
```

1.18. Developing Resource Writers

A Resource Writer 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 they need to store a `Resource`.

1.18.1. Create a New `ResourceWriter`

Complete the following procedure to create a `ResourceWriter`.

1. Create a Java class that implements the `DDF.catalog.resource.ResourceWriter` interface.

ResourceWriter Implementation Skeleton

```
import java.io.IOException;
import java.net.URI;
import java.util.Map;
import DDF.catalog.resource.Resource;
import DDF.catalog.resource.ResourceNotFoundException;
import DDF.catalog.resource.ResourceNotSupportedException;
import DDF.catalog.resource.ResourceWriter;

public class SampleResourceWriter implements ResourceWriter {

    @Override
    public void deleteResource(URI uri, Map<String, Object> arguments) throws
```

```
ResourceNotFoundException, IOException {
       // WRITE IMPLEMENTATION
     }

    @Override
    public URI storeResource(Resource resource, Map<String, Object> arguments)throws
ResourceNotSupportedException, IOException {
       // WRITE IMPLEMENTATION
       return null;
    }

    @Override
    public URI storeResource(Resource resource, String id, Map<String, Object> arguments)
throws ResourceNotSupportedException, IOException {
       // WRITE IMPLEMENTATION
       return null;
    }

}
```

1. Register the implementation as a Service in the OSGi Service Registry.

Blueprint Service Registration Example

```
...
<service ref="ResourceWriterReference" interface="DDF.catalog.resource.ResourceWriter" />
...
```

1. Deploy the OSGi bundled packaged service to the DDF run-time (Refer to the OSGi Basics - Bundles section.)

TIP	**ResourceWriter Javadoc** Refer to the Catalog API Javadoc for more information about the methods required for implementing the interface.

1.19. Developing Filters

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

NOTE	Refer to the GeoTools javadoc for more information on `FilterFactoryImpl`.
WARNING	Implementing the Filter interface directly is only for extremely advanced use

> cases and is highly discouraged. Instead, use of the DDF-specific `FilterBuilder` API is recommended.

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

Categories of Filters

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

1.19.1. Units of Measure

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

- FOOT
- METRE
- PIXEL

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

1.19.2. Filter Examples

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

Simple Contextual Search

```
org.geotools.api.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.geotools.api.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.

Absolute Temporal Search

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

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

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

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

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

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

1.19.2.1. 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).

Table 7. `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.

Creating the search phrase "mission and planning"

```
org.geotools.api.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 );
```

1.19.2.1.1. Tree View of Creating Filters

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

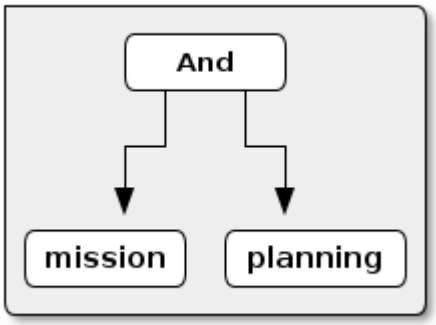

Figure 4. Filter Example Tree Diagram

1.19.2.1.2. XML View of Creating Filters

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

Pseudo XML of Example Creating-Filters-3

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

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

1.19.2.2. Fuzzy Operations

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

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

boolean isCaseSensitive = false ;

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

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

1.19.3. Parsing Filters

According to the OGC Filter Specification 04-095: a "(filter expression) representation can be ... parsed and then transformed into whatever target language is required to retrieve or modify object instances stored in some persistent object store." Filters can be thought of as the `WHERE` clause for a SQL SELECT statement to "fetch data stored in a SQL-based relational database."

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

Alternately, `org.geotools.api.filter.Filter` implementations can be parsed using implementations of the interface `org.geotools.api.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` may be needed to support `Filter` functionality not currently handled by the `FilterAdapter` and `FilterDelegate` reference implementation.

1.19.3.1. 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.

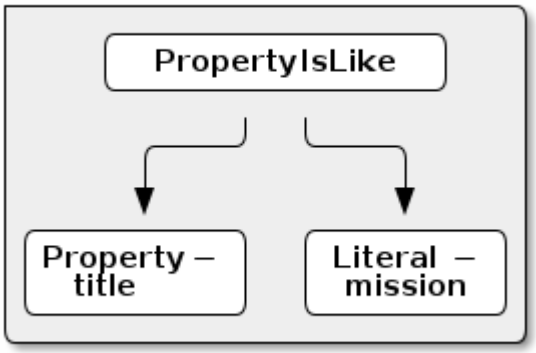

Figure 5. Parsing Filters Tree Diagram

1.19.3.2. Interpreting a Filter to Create XQuery

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

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

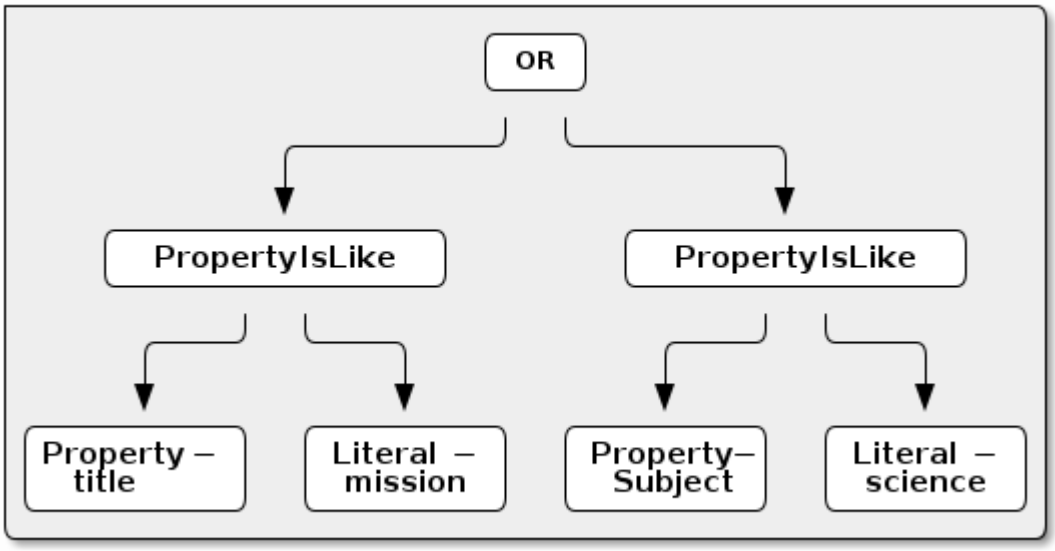

Figure 6. Parsing Filters XQuery

1.19.3.2.1. FilterAdapter/Delegate Process for Figure Parsing

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` calls 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` calls 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 is returned from the `FilterAdapter` adapt method.

1.19.3.2.2. FilterVisitor Process for Figure Parsing

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;
...
}
```

1.19.4. Filter Profile

The filter profile maps filters to metacard types.

1.19.4.1. 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.

1.19.4.2. Catalog Filter Profile

The following table displays the common metacard attributes with their respective types for reference.

Table 8. Metacard Attribute To Type Mapping

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

1.19.4.2.1. 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 are not supported by standard sources. The following table maps the metacard types to supported comparison operators.

Table 9. Metacard Types to Comparison Operators

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

Table 10. Comparison Operators

Operator	Description
PropertyIsBetween	Lower ⇐ Property ⇐ Upper
PropertyIsEqualTo	Property == Literal
PropertyIsGreaterThan	Property > Literal
PropertyIsGreaterThanOrEqualTo	Property >= Literal
PropertyIsLessThan	Property < Literal

Operator	Description
PropertyIsLessThanOrEqualTo	Property ⇐ Literal
PropertyIsLike	Property LIKE Literal Equivalent to SQL "like"
PropertyIsNotEqualTo	Property != Literal
PropertyIsNull	Property == null

1.19.4.2.2. Logical Operators

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

Table 11. Supported Logical Operators

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

1.19.4.2.3. Temporal Operators

Temporal operators compare a date associated with a property name to a given Literal date or date range.

Table 12. Supported Temporal Operators

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

Literal values can be either date instants or date periods.

Table 13. Temporal Operator Descriptions

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

1.19.4.2.4. Spatial Operators

Spatial operators compare a geometry associated with a property name to a given Literal geometry.

Table 14. Supported Spatial Operators.

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

Geometries are usually represented as Well-Known Text (*WKT*).

Table 15. Spatial Operator Descriptions

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

1.20. Developing Filter Delegates

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.

1.20.1. Creating a New Filter Delegate

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

1.20.1.1. Implementing the Filter Delegate

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

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

1.20.1.2. Throwing Exceptions

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

1.20.1.3. Using the Filter Adapter

The FilterAdapter can be requested from the OSGi registry.

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

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

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

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

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

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

Import the 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`

1.20.1.4. 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 * the maximum dimension of the intersection value includes points interior to both the geometries * the intersection value is not equal to either of the geometries.
`Disjoint`	True if the two geometries do not touch or intersect.
`DWithin`	True if the geometry being tested is within the stated distance of the geometry provided.
`Intersects`	True if the two geometries intersect. This is a convenience method as `Not Disjoint(A,B)` gets the same result.

Spatial	Definition
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

1.21. Developing Query Options

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

`QueryImpl` *Example*

```
/*
  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);
```

1.21.1. Evaluating a query

Every Source must be able to evaluate a Query object. Nevertheless, each Source could evaluate the Query differently depending on what that Source supports as to properties and query capabilities. For instance, a common property all Sources understand is id, but a Source could possibly store frequency values under the property name "frequency." Some Sources may not support frequency property inquiries and 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.

Table 16. 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. If a zero or negative timeout is set, the catalog framework defaults to a value configurable via the Admin UI under Catalog → Configuration → Query Operations.

1.21.2. Commons-DDF Utilities

The `commons-DDF` bundle provides utilities and functionality commonly used across other DDF components, such as the endpoints and providers.

1.21.2.1. FuzzyFunction

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

1.21.2.2. 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.

1.22. Developing Action Components

To provide a service, such as a link to a metacard, the `ActionProvider` interface should be implemented. An `ActionProvider` essentially provides a List of `Actions` when given input that it can recognize and handle. For instance, if a REST endpoint ActionProvider was given a metacard, it could provide a link based on the metacard's ID. An Action Provider performs an action when given a subject that it understands. If it does not understand the subject or does not know how to handle the given input, it returns `Collections.emptyList()`. An Action Provider is required to have an ActionProvider id. The Action Provider must register itself in the OSGi Service Registry with the `ddf.action.ActionProvider` interface and must also have a service property value for `id`. An action is a URL that, when invoked, provides a resource or executes intended business logic.

1.22.1. Action Component Naming Convention

For each Action, a title and description should be provided to describe what the action does. The recommended naming convention is to use the verb 'Get' when retrieving a portion of a metacard, such as the metadata or thumbnail, or when downloading a resource. The verb 'Export' or the expression 'Export as' is recommended when the metacard is being exported in a different format or presented after going some transformation.

1.22.1.1. Action Component Taxonomy

An Action Provider registers an `id` as a service property in the OGSi Service Registry based on the type of service or action that is provided. Regardless of implementation, if more than one Action Provider provides the same service, such as providing a URL to a thumbnail for a given metacard, they must both register under the same `id`. Therefore, Action Provider implementers must follow an Action Taxonomy.

The following is a sample taxonomy:

1. `catalog.data.metacard` shall be the grouping that represents Actions on a Catalog metacard.
    a. `catalog.data.metacard.view`
    b. `catalog.data.metacard.thumbnail`
    c. `catalog.data.metacard.html`
    d. `catalog.data.metacard.resource`
    e. `catalog.data.metacard.metadata`

Table 17. Action ID Service Descriptions

ID	Required Action	Naming Convention
`catalog.data.metacard.view`	Provides a valid URL to view a metacard. Format of data is not specified, so the representation can be in XML, JSON, or other.	Export as ...
`catalog.data.metacard.thumbnail`	Provides a valid URL to the bytes of a thumbnail (`Metacard.THUMBNAIL`) with MIME type image/jpeg.	Export as Thumbnail
`catalog.data.metacard.map.overlay.thumbnail`	Provides a metacard URL that translates the metacard into a geographically aligned image (suitable for overlaying on a map).	Export as Thumbnail Overlay
`catalog.data.metacard.html`	Provides a valid URL that, when invoked, provides an HTML representation of the metacard.	Export as HTML
`catalog.data.metacard.xml`	Provides a valid URL that, when invoked, provides an XML representation of the metacard.	Export as XML
`catalog.data.metacard.geojson`	Provides a valid URL that, when invoked, provides an XML representation of the metacard.	Export as GeoJSON
`catalog.data.metacard.resource`	Provides a valid URL that, when invoked, provides the underlying resource of the metacard.	Export as Resource
`catalog.data.metacard.metadata`	Provides a valid URL to the XML metadata in the metacard (`Metacard.METADATA`).	Export as Metadata

1.23. Configuring Managed Service Factory Bundles

Services that are created using a Managed Service Factory can be configured using `.config` files as well. These configuration files, however, follow a different naming convention than `.cfg` files. The filenames must start with the Managed Service Factory PID, be followed by a dash and a unique identifier, and have a `.config` extension. For instance, assuming that the Managed Service Factory PID is `org.codice.ddf.factory.pid` and two instances of the service need to be configured, files `org.codice.ddf.factory.pid-<UNIQUE ID 1>.config` and `org.codice.ddf.factory.pid-<UNIQUE ID 2>.config` should be created and added to `<DDF_HOME>/etc`.

The unique identifiers used in the file names have no impact on the order in which the configuration files are processed. No specific processing order should be assumed. Also, a new service is created and configured every time a configuration file matching the Managed Service Factory PID is added to the directory, regardless of the *unique id* used.

Any `service.factoryPid` and `service.pid` values in these `.config` files are overridden by the values parsed from the file name, so `.config` files should not contain these properties.

1.23.1. File Format

The basic syntax of the `.config` configuration files is similar to the older `.cfg` files but introduces support for lists and types other than simple strings. The type associated with a property must match the type attribute used in the corresponding `metatype.xml` file when applicable.

The following table shows the format to use for each property type supported.

Table 18. Property Formats

Type	Format (see details below for variations)	Example
String	`name="value"`	`name="John"`
Boolean	`name=B"true|false"`	`authorized=B"true"`
Integer	`name=I"value"`	`timeout=I"10"`
Long	`name=L"value"`	`diameter=L"100"`
Float	`name=F"value"`	`cost=F"1093140480"`
Double	`name=D"value"`	`latitude=D"4636745974857667812"`

List of Strings	`name=["value1","value2",…]`	`complexStringArray=[ \`
		`  "{\"url\"\ \"http://test.sample.com\"\`
		`\"layers\"\ [\"0\"]\ \"VERSION\"\`
		`\"1.1|1.2\"\ \"image/png\"}\ \"beta\"\`
		`1}", \`
		`  "{\"url\"\ \"http://test.sample.com"\`
		`0.5}", \`
		`  "/security-config\=SAML|basic", \`
		`  ]`

List of Booleans	`name=B["true|false","true|false",…]`	`authorizedList=B[ \`
		`  "true", \`
		`  "false", \`
		`  ]`

List of Integers	`name=I["value1","value2",…]`	`sizes=I[ \`
		`  "10", \`
		`  "20", \`
		`  "30", \`
		`  ]`

Type	Format (see details below for variations)	Example

List of Longs	`name=L["value1","value2",…]`	`sizes=L[ \`
		`  "100", \`
		`  "200", \`
		`  "300", \`
		`  ]`

List of Floats	`name=F["value1","value2",…]`	`sizes=F[ \`
		`  "1066192077", \`
		`  "1074580685", \`
		`  "1079194419", \`
		`  ]`

List of Doubles	`name=D["value1","value2",…]`	`sizes=D[ \`
		`  "4607736361554183979", \`
		`  "4612212939583790252", \`
		`  "4614714689176794563", \`
		`  ]`

NOTE

- Values with types other than String must be prefixed with a lower-case or upper-case character. See the examples in the table.
  - Boolean: `B` or `b`
  - Integer: `I` or `i`
  - Long: `L` or `l`
  - Float: `F` or `f`
  - Double: `D` or `d`
- Equal signs (`=`), double quotes (`"`), and spaces within values must must be escaped using a backslash (`\`).
- When properties are split over multiple lines for readability, end of lines must be specified with a backslash (`\`). See the examples for lists in the table.
- A comma (`,`) after the last value in a list is optional.
- Surrounding the equal signs (`=`) with spaces for properties is optional. Because there is a known issue when using OPS4J Pax Exam 4.11.0 and modifying `.config` files that include spaces, all default `.config` files that may be modified in OPS4J Pax Exam 4.11.0 tests should not include spaces.
- Boolean values default to `false` if any value other than `true` is provided.

- Float values must be representated in the IEEE 754 floating-point "single format" bit layout, preserving Not-a-Number (NaN) values. For example, `F"1093140480"` corresponds to `F"10.5"`. See the documentation for `java.lang.Integer#parseInt(java.lang.String)` and `java.lang.Float#intBitsToFloat(int)` for more details.
- Double values must be represented in the IEEE 754 floating-point "double format" bit layout, preserving Not-a-Number (NaN) values. For example, `D"4636745974857667812"` corresponds to `D"100.1234"`. See the documentation for `java.lang.Long#parseLong(java.lang.String)` and `java.lang.Double#longBitsToDouble` for more details.

Sample configuration file

```
webAuthenticationTypes="basic"
endpointAuthenticationTypes=""
sessionAccess=B"true"
guestAccess=B"true"
realms=[ \
  "/\=karaf", \
  ]
requiredAttributes=[ \
  "/\=", \
  "/admin\={http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role\=admin}", \
  "/system\={http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role\=admin}", \
  "/security-
config\={http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role\=admin}", \
  ]
whiteListContexts=[ \
  "/services/SecurityTokenService", \
  "/services/internal/metrics", \
  "/services/saml", \
  "/proxy", \
  "/services/csw", \
  ]
```

1.24. Developing XACML Policies

This document assumes familiarity with the XACML schema and does not go into detail on the XACML language. When creating a policy, a target is used to indicate that a certain action should be run only for one type of request. Targets can be used on both the main policy element and any individual rules. Targets are geared toward the actions that are set in the request. These actions generally consist of the standard CRUD operations (create, read, update, delete) or a SOAPAction if the request is coming through a SOAP endpoint.

NOTE	These are only the action values that are currently created by the components that come with DDF. Additional components can be created and added to DDF to identify specific actions.

In the examples below, the policy has specified targets for the above type of calls. For the Filtering code, the target was set for `filter`, and the Service validation code targets were geared toward two services: `query` and `LocalSiteName`. In a production environment, these actions for service authorization generally are full URNs that are described within the SOAP WSDL.

1.24.1. XACML Policy Attributes

Attributes for the XACML request are populated with the information in the calling subject and the resource being checked.

1.24.2. XACML Policy Subject

The attributes for the subject are obtained from the SAML claims and populated within the XACML policy as individual attributes under the `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject` category. The name of the claim is used for the `AttributeId` value. Examples of the items being populated are available at the end of this page.

1.24.3. XACML Policy Resource

The attributes for resources are obtained through the permissions process. When checking permissions, the XACML processing engine retrieves a list of permissions that should be checked against the subject. These permissions are populated outside of the engine and should be populated with the attributes that should be asserted against the subject. 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.

1.24.4. Using a XACML Policy

To use a XACML policy, copy the XACML policy into the `<DDF_HOME>/etc/pdp/policies` directory.

1.25. Assuring Authenticity of Bundles and Applications

DDF Artifacts in the JAR file format (such as bundles or KAR files) can be signed and verified using the tools included as part of the Java Runtime Environment.

1.25.1. Prerequisites

To work with Java signatures, a keystore/truststore is required. For testing or trial purposes DDF can sign and validate using a self-signed certificate, generated with the keytool utility. In an actual installation, a certificate issued from a trusted Certificate Authority is used.

Additional documentation on keytool can be found at Keytool home.

Using keytool to generate a self-signed certificate keystore

```
~ $ keytool -genkey -keyalg RSA -alias selfsigned -keystore keystore.jks -storepass
password -validity 360 -keysize 2048
What is your first and last name?
  [Unknown]:  Nick Fury
What is the name of your organizational unit?
  [Unknown]:  Marvel
What is the name of your organization?
  [Unknown]:  SHIELD
What is the name of your City or Locality?
  [Unknown]:  New York
What is the name of your State or Province?
  [Unknown]:  NY
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Nick Fury, OU=SHIELD, O=Marvel, L="New  York", ST=NY, C=US correct?
  [no]:  yes
Enter key password for <selfsigned>
    (RETURN if same as keystore password):
Re-enter new password:
```

1.25.2. Signing a JAR/KAR

Once a keystore is available, the JAR can be signed using the `jarsigner` tool.

Additional documentation on jarsigner can be found at Jarsigner.

Using jarsigner to sign a KAR

```
~ $ jarsigner -keystore keystore.jks -keypass shield -storepass password  catalog-app-
2.5.1.kar selfsigned
```

1.25.2.1. Verifying a JAR/KAR

The jarsigner utility is also used to verify a signature in a JAR-formatted file.

Using jarsigner to verify a file

```
~ $ jarsigner -verify -verbose -keystore keystore.jks catalog-app-2.5.1.kar
        9447 Mon Oct 06 17:05:46 MST 2014 META-INF/MANIFEST.MF
        9503 Mon Oct 06 17:05:46 MST 2014 META-INF/SELFSIGN.SF

[... section abbreviated for space]
```

```
smk     6768 Wed Sep 17 17:13:58 MST 2014 repository/ddf/catalog/security/catalog-
security-logging/2.5.1/catalog-security-logging-2.5.1.jar
  s = signature was verified
  m = entry is listed in manifest
  k = at least one certificate was found in keystore
  i = at least one certificate was found in identity scope
jar verified.
```

Note the last line: *jar verified*. This indicates that the signatures used to sign the JAR (or in this case, KAR) were valid according to the trust relationships specified by the keystore.

1.26. WFS Services

The Web Feature Service (WFS) is an Open Geospatial Consortium (OGC) Specification. DDF supports the ability to integrate WFS 1.1, and 2.0 Web Services.

NOTE	DDF does not include a supported WFS Web Service (Endpoint) implementation. Therefore, federation for 2 DDF instances is not possible via WFS.

WFS Features

When a query is issued to a WFS server, the output of the query is an XML document that contains a collection of feature member elements. Each WFS server can have one or more feature types with each type being defined by a schema that extends the WFS `featureMember` schema. The schema for each type can be discovered by issuing a `DescribeFeatureType` request to the WFS server for the feature type in question. The WFS source handles WFS capability discovery and requests for feature type description when an instance of the WFS source is configured and created.

See the WFS v1.1.0 Source, or WFS v2.0.0 Source for more information about how to configure a WFS source.

Converting a WFS Feature

In order to expose WFS features to DDF clients, the WFS feature must be converted into the common data format of the DDF, a metacard. The OGC package contains a `GenericFeatureConverter` that attempts to populate mandatory metacard fields with properties from the WFS feature XML. All properties are mapped directly to new attributes in the metacard. However, the `GenericFeatureConverter` may not be able to populate the default metacard fields with properties from the feature XML.

Creating a Custom Converter

To more accurately map WFS feature properties to fields in the metacard, a custom converter can be created. The OGC package contains an interface, `FeatureConverter`, which extends the http://xstream.codehaus.org/javadoc/com/thoughtworks/xstream/converters/Converter.htmlConverter interface provided by the XStream project. XStream is an open source API for serializing XML into Java objects and vice-versa. Additionally, a base class, `AbstractFeatureConverter`, has been created to handle the mapping of many fields to reduce code duplication in the custom converter classes.

1. Create the `CustomConverter` class extending the `ogc.catalog.common.converter.AbstractFeatureConverter` class.

```
public class CustomConverter extends ogc.catalog.common.converter
.AbstractFeatureConverter
```

2. Implement the `FeatureConverterFactory` interface and the `createConverter()` method for the `CustomConverter`.

```
public class CustomConverterFactory implements FeatureConverterFactory {
    private final featureType;
    public CustomConverterFactory(String featureType) {
        this.featureType = featureType;
    }
    public FeatureConverter createConverter() {
        return new CustomConverter();
    }
    public String getFeatureType() {
        return featureType;
    }
}
```

3. Implement the `unmarshal` method required by the `FeatureConverter` interface. The `createMetacardFromFeature(reader, metacardType)` method implemented in the `AbstractFeatureConverter` is recommended.

```
public Metacard unmarshal(HierarchicalStreamReader reader, UnmarshallingContext ctx) {
  MetacardImpl mc = createMetacardFromFeature(reader, metacardType);
  //set your feature specific fields on the metacard object here
  //
  //if you want to map a property called "beginningDate" to the Metacard.createdDate
field
  //you would do:
  mc.setCreatedDate(mc.getAttribute("beginningDate").getValue());
}
```

4. Export the `ConverterFactory` to the OSGi registry by creating a `blueprint.xml` file for its bundle. The bean id and argument value must match the WFS Feature type being converted.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" xmlns:cm=
"http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.1.0">
  <bean id="custom_type" class="com.example.converter.factory.CustomConverterFactory">
```

```
    <argument value="custom_type"/>
  </bean>
  <service ref="custom_type" interface=
"ogc.catalog.common.converter.factory.FeatureConverterFactory"/>
</blueprint>
```

1.27. JSON Definition Files

DDF supports adding new attribute types, metacard types, validators, and more using json-formatted definition files.

The following may be defined in a JSON definition file:

- Attribute Types
- Metacard Types
- Global Attribute Validators
- Metacard Validators
- Default Attribute Values
- Attribute Injections

1.27.1. Definition File Format

A definition file follows the JSON format as specified in ECMA-404. All definition files must be valid JSON in order to be parsed.

A single definition file may define as many of the types as needed. This means that types can be defined across multiple files for grouping or clarity.

1.27.2. Deploying Definition Files

The file must have a `.json` extension in order to be picked up by the deployer. Once the definition file is ready to be deployed, put the definition file `<filename>.json` into the `etc/definitions` folder.

Definition files can be added, updated, and/or deleted in the `etc/definitions` folder. The changes are applied dynamically and no restart is required.

If a definition file is removed from the `etc/definitions` folder, the changes that were applied by that file are undone.

1.28. Developing Subscriptions

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

1.28.1. Subscription Lifecycle

A Subscription itself is a series of events during which various plugins or transformers can be called to process the subscription.

1.28.1.1. Creation

- Subscriptions are created directly with theEvent Processor or declaratively through use of the Whiteboard Design Pattern.
- The Event Processor invokes each Pre-Subscription Plugin and, if the subscription is not rejected, the subscription is activated.

1.28.1.2. Evaluation

- When a metacard matching the subscription is created, updated, or deleted in any Source, each Pre-Delivery Plugin is invoked.
- If the delivery is not rejected, the associated Delivery Method callback is invoked.

1.28.1.3. 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.

1.28.1.4. Durability

Subscription durability is not provided by the Event Processor. Thus, all subscriptions are transient and not recreated in the event of a system restart.

It is the responsibility of Endpoints using subscriptions to persist and re-establish the subscription on startup.

This decision was made for the sake of simplicity, flexibility, and the inability of the Event Processor to recreate a fully configured Delivery Method without being overly restrictive.

IMPORTANT	**Subscriptions are not persisted by the Catalog itself.** Subscriptions must be explicitly persisted by an endpoint and are not persisted by the Catalog. The Catalog Framework, or more specifically the Event Processor itself, does not persist subscriptions. Certain endpoints, however, can persist the subscriptions on their own and recreate them on system startup.

1.28.2. Creating a Subscription

Currently, the Catalog reference implementation does not contain a subscription endpoint. Therefore, an endpoint that exposes a web service interface to create, update, and delete subscriptions would provide a client's subscription filtering criteria to be used by Catalog's Event Processor to determine which events are of interest to the client. The endpoint client also provides the callback URL of the

event consumer to be called when an event matching the subscription's criteria is found. This callback to the event consumer is made by a Delivery Method implementation that the client provides when the subscription is created. Whenever an event occurs in the Catalog matching the subscription, the Delivery Method implementation is called by the Event Processor. The Delivery Method, in turn, sends 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.

1.28.2.1. Event Processing and Notification

If an event matches a subscription's criteria, any pre-delivery plugins that are installed are invoked, the subscription's `DeliveryMethod` is retrieved, and its operation corresponding to the type of ingest event is invoked. For example, the `DeliveryMethod created()` function is called when a metacard is created. The `DeliveryMethod` operations subsequently invoke the corresponding operation in the client's event consumer service, which is specified by the callback URL provided when the `DeliveryMethod` was created. 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.

The Standard Event Processor is an implementation of the Event Processor and provides the ability to create/delete subscriptions. Events are generated by the CatalogFramework as metacards are created/updated/deleted and the Standard Event Processor is called since it is also a Post-Ingest Plugin. The Standard Event Processor checks each event against each subscription's criteria.

When an event matches a subscription's criteria, the Standard Event Processor invokes:

- each pre-delivery plugin on the metacard in the event.
- the `DeliveryMethod` operation corresponding to the type of event being processed, such as the `created()` operation for the creation of a metacard.

Available Event Processor

- Standard Event Processor

1.28.2.1.1. Using DDF Implementation

If applicable, the implementation of `Subscription` that comes with DDF should be used. It is available at `ddf.catalog.event.impl.SubscriptionImpl` and offers a constructor that takes in all of the necessary objects. Specifically, all that is needed is a `Filter`, `DeliveryMethod`, `Set<String>` of source IDs, and a `boolean` for enterprise.

The following is an example code stub showing how to create a new instance of Subscription using the DDF implementation.

Creating a Subscription

```
// Create a new filter using an imported FilterBuilder
Filter filter = filterBuilder.attribute(Metacard.ANY_TEXT).like().text("*");

// Create a implementation of DeliveryMethod
DeliveryMethod deliveryMethod = new MyCustomDeliveryMethod();

// Create a set of source ids
// This set is empty as the subscription is not specific to any sources
Set<String> sourceIds = new HashSet<String>();

// Set the isEnterprise boolean value
// This subscription example should notifications from all sources (not just local)
boolean isEnterprise = true;

Subscription subscription = new SubscriptionImpl(filter, deliveryMethod, sourceIds
,isEnterprise);
```

1.28.2.2. Delivery Method

A Delivery Method provides the operation (created, updated, deleted) for how an event's metacard can be delivered.

A Delivery Method is associated with a subscription and contains the callback URL of the event consumer to be notified of events. The Delivery Method encapsulates the operations to be invoked by the Event Processor when an event matches the criteria for the subscription. The Delivery Method's operations are responsible for invoking the corresponding operations on the event consumer associated with the callback URL.

1.29. Contributing to Documentation

DDF documentation is included in the source code, so it is edited and maintained in much the same way.

`src/main/resources`

Table 19. Documentation Directory Structure and Contents

Directory	Contents
`content`	Asciidoctor-formatted files containing documentation contents and the header information needed to organize them.
`images`	Screenshots, icons, and other image files used in documentation.
`templates`	Template files used to compile the documentation for display.
`jbake.properties`	Properties file defining content types and other parameters.

1.29.1. Editing Existing Documentation

Update existing content when code behavior changes, new capabilities are added to features, or the configuration process changes. Content is organized within the `content` directory in sub directories according to the audience and purpose for each document in the documentation library. Use this list to determine placement of new content.

Documentation Sections

Introduction/Core Concepts

This section is intended to be a high-level, executive summary of the features and capabilities of DDF. Content here should be written at a non-technical level.

Quick Start

This section is intended for getting set up with a test, demonstration, or trial instance of DDF. This is the place for non-production shortcuts or workarounds that would not be used in a secured, hardened installation.

Managing

The managing section covers "how-to" instructions to be used to install, configure, and maintain an instance of DDF in a production environment. This content should be aimed at system administrators. Security hardening should be integrated into these sections.

Using

This section is primarily aimed at the final end users who perform tasks with DDF. This content should guide users through common tasks and user interfaces.

Integrating

This section guides developers building other projects looking to connect to new or existing instances of DDF.

Developing

This section provides guidance and best practices on developing custom implementations of DDF components, especially ones that may be contributed into the code baseline.

Architecture

This section is a detailed description of the architectural design of DDF and how components work together.

Reference

This section is a comprehensive list of features and possible configurations.

Metadata Reference

This section details how metadata is extracted and normalized by DDF.

Documentation

This is a collection of all of the individual documentation pages in one html or pdf file.

See the style guide for more guidance on stylistic and formatting concerns.

1.29.2. Adding New Documentation Content

If creating a new section is required, there are some minimal requirements for a new `.adoc` file.

Header content

The templates scan the header information to place it into the correct place within the documentation. Different sections have different headers required, but some common attributes are always required.

- `type`: roughly maps to the section or subSection of the documentation.
- `title`: title of the section or subsection contained in the file.
- `status`: set to `published` to include within the documentation, set to `draft` to hide a work-in-progress section.
- `order`: used in sections where order needs to be enforced.
- `summary`: brief summary of section contents. Some, but not all, summaries are included by templates.

1.29.3. Creating a New Documentation Template

To create a new, standalone documentation page, create a new template in the `templates` directory. Optionally, this template can `include` some of the internal templates in the `templates/build` directory, but this is not required.

For guidance on using the freemarker syntax, see the Freemarker documentation.

1.29.4. Extending Documentation in Downstream Distributions

By mimicking the build and directory structure of the documentation, downstream projects are able to leverage the existing documentation and insert content before and after sections of the DDF documentation.

Documentation Module Directory Structure

```
-docs
  -src
    -main
      -resources
        -content
        -images
        -templates
```

`content`

Contains the .adoc files that make up the content. Sub-directories are organized according to the documents that make up the main library.

`images`

any pre-existing images, such as screenshots, to be included in the documentation.

`templates`

template files used to create documentation artifacts. A `build` sub-directory holds the templates that are not standalone documents to render specific sections.

2. Development Guidelines

2.1. Contributing

The Distributed Data Framework is free and open-source software offered under the GNU Lesser General Public License. The DDF is managed under the guidance of the Codice Foundation. Contributions are welcomed and encouraged. Please visit the Codice DDF Contributor Guidelines and the DDF source code repository for more information.

2.2. OSGi Basics

DDF runs on top of an OSGi framework, a Java virtual machine (JVM), several choices of operating systems, and the physical hardware infrastructure. The items within the dotted line represent the standard DDF components.

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).

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

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

Capability

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

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.

Application

A JSON file defining a collection of bundles with configurations to be displayed in the Admin Console.

2.2.1. Packaging Capabilities as Bundles

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

The best resource for learning about the structure and headers in the manifest definition is in section 3.6 of the OSGi Core Specification. The bundles within DDF are created using the maven bundle plug-in, which uses the BND tool.

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

2.2.1.1. Creating a Bundle

2.2.1.1.1. Bundle Development Recommendations

Avoid creating bundles by hand or editing a manifest file

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

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

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

Embedding is an implementation detail

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

Bundles should never be embedded

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

Bundles should expose service implementations

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

Bundles should generally *only* export service packages

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

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

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

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

2.2.1.1.2. 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>
...
```

2.2.1.2. Third Party and Utility Bundles

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

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

2.2.1.3. Deploying a Bundle

A bundle is typically installed in one of two ways:

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

The fastest way to deploy a created bundle during development is to copy it to the `/deploy` directory of a running DDF. This directory checks for new bundles and deploys them immediately. According to Karaf documentation, "Karaf supports hot deployment of OSGi bundles by monitoring JAR files inside the `[home]/deploy` directory. Each time a JAR is copied in this folder, it is installed inside the runtime. It can be updated or deleted and changes are 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.

2.2.1.4. Verifying Bundle State

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

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

quickly find the bundle.

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

Verifying with grep

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

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

2.3. High Availability Guidance

Capabilities that need to function in a Highly Available Cluster should have one of the two below properties.

Stateless

Stateless capabilities function in a Highly Available Cluster because no synchronization between DDF nodes is necessary.

Common storage

If a capability must store data or share state with another node, then the data or shared state must be accessible to all nodes in the Highly Available Cluster. For example, the Catalog's storage provider must be accessible to all DDF nodes.

Index

A

Attribute injections, 20
Attribute Type Definition File, 17

B

Bundles, 104

C

Catalog API implementations, 6
Catalog Plugins, 34
Catalog Provider, 31
Catalog Store, 31
Common storage, 107
Connected Source, 31
Contributions, 103
Custom endpoints, 23

D

Default Attribute Values, 19

F

Federated Source, 31
Filter, 72
Filter Delegates, 83

G

Global Attribute Validators, 14

I

Input Transformers, 24

J

JSON Definition Files, 97

M

Metacard Transformers, 28
Metacard Type definition files, 10
Metacard Validator definitions, 13
Metacard Validators, 13

O

OSGi, 103

Q

Query, 86
Query Response Transformers, 29

R

Registry Clients, 68
Resource Reader, 68
Resource Writer, 71

S

Sources, 30
Stateless, 107
STS Claims Handlers, 41
Subscriptions, 97

T

Token validators, 40

X

XACML Policies, 92