OPeNDAP, Inc.

2025-04-19 02:05:07 UTC

Table of Contents

OPeNDAP Lightweight Frontend Service (olfs)

See the rendered olfs document here.

Chapter 1. Overview

The OPeNDAP Lightweight Frontend Service (***olfs***) is the public/outward facing component of the Hyrax server.

Chapter 2. What does it Do?

The ***olfs*** provides a number of interfaces through which users and programmatic clients can request and subset data.

2.1. Supported APIs:

- Aggregation Service API (*Does this even work? When was the last time we tried it?*)
- DAP2 data access API
- DAP4 data access API
- WCS 2.0 API
- w10n data access api
- THREDDS Client-Side Catalog API for machine traversable catalog/inventories.
- **File Download Service** - When this option is enabled the clients are allowed to download the underlying data files directly from the Hyrax service. Not super efficient use of Hyrax t.b.h. some (all?) of the on-prem NASA DAACs operate a separate download service for the data files.
- **Gateway service** - This service allows the user to ask Hyrax to retrieve a remote dataset file and then provide DAP services for that file.
- **NGAP DAP service** - The NGAP DAP service is a NASA specific service that integrates into NASA's NGAP service chain, providing a publicly accessible DAP service endpoint for their cloud deployed data holds. The NGAP DAP service utilizes the NASA Common Metadata Repository (CMR), Thin Egress Client (TEA), and AWS S3 to provide access to data using precomputed dmr++ files and HTTP Range GET commands to extract data values from files/objects held in S3. Note: The NGAP DAP service does not provide a traversable catalog/inventory of the data holdings.

2.2. Supported User Interfaces

- **Data Request Form (aka the *Interface From Hell*, aka the *IFH*)**: Both the DAP2 and DAP4 services provide a web UI that allows a user to build and execute data access requests. IFH pages are built by retrieving the metadata for the dataset from the BES as an XML document and then using an XSL transform to produce the IFH response from the metadata document.
  - For the DAP2 IFH a DDX (an XML encoding of DAP2 metadata) is used to build the form.
  - For the DAP4 IFH the DMR document is used.
  - Each case has its own XSLT which is conditioned by the ***olfs*** prior to the transformation step.
- **Browser Navigable Catalog**: The OLFS uses BES showNode commands to discover what the BES "catalog" (usually a filesystem but maybe not) contains. The BES responses are XML documents that the OLFS operates on with an XSL transform to produce the Hyrax "blue bar" browser navigation pages. The server will try to return this page for any url path ending in `/` or in `/contents.html` Some parts of the catalog ui representation are controlled by the ***olfs.xml*** file used with the deployment. (See Configuration)

- **THREDDS Catalogs, a machine-readable catalog** *No Scraping Required!* The OLFS automatically provides a machine-readable THREDDS catalog for the inventory in its catalog. In fact, it uses the same inputs and only differs in the XSL from the browser navigable catalog described above to produce a THREDDS catalog response. (See [browser_catalog])

Chapter 3. How is the runtime code assembled?

The primary code pattern used by the ***olfs*** for implementing services is to subclass *HttpServlet* and create specialized implementations that utilize one or more *DispatchHandler* implementations to claim requests and respond to them. In some cases a servlet might utilize a single instance of *DispatchHandler* to run the show. For example, the servlet *opendap.build_dmrpp.BuildDmrppServlet* relies solely on *opendap.build_dmrpp.BuildDmrppDispatchHandler* and the servlet *opendap.wcs.v2.0.Servlet* relies solely on *opendap.wcs.v2.0.HttpGetHandler*. In other examples there are many *DispatchHandler* objects used by the servlet implementation. For example *opendap.coreServlet.DispatchServlet* uses about a dozen different *DispatchHandler* instances.

Chapter 4. How? How does it work?

It's complicated, so first I'll provide a simplified narrative of the request/response operation, and then I'll add a couple of sections regarding other parts of the operation.

4.1. DAP Request/Response

Let's look at how the primary servlet, *opendap.coreServlet.DispatchServlet*, handles an incoming DAP service request:

In the *opendap.coreServlet.DispatchServlet.doGet()* method the code first handles some bookkeeping (such as producing a request id and starting the access log for the request) before iterating over an internal vector of *DispatchHandler* objects, asking each in turn if it can service the request. The first *DispatchHandler* to claim the request is then asked to service it and transmit the response. The order of the *DispatchHandler* objects in the vector is critical, as there are greedy *DispatchHandler* implementations (*FileDispatchHandler* for example) that will claim things that should/would have been claimed by the intended *DispatchHandler* if asked prior. The order is established in the *DispatchServlet.init()* method.

If the request is for some type of DAP response then the *BesDapDispatcher* will claim it.

Inside *BesDapDispatcher* there is a vector of *Dap4Responder* objects, each one associated with a particular DAP response type (such as: use DAP2 and return a netcdf-3 encoded response, use DAP4 and return the DMR response, etc.) The *BesDapDispatcher* iterates over its vector of *Dap4Responder* objects and hands the response duties to the first one that claims the request.

One might ask: How does all this "claiming" work? *Through the magic of regular expressions!* Each of the *Dap4Responder* objects contains a primary regular expression which is used to match a URL path in the request. For example the *NormativeDMR* responder has something like this `^.*\.dmr$`. But it's more complex because in DAP4 there are several different alternate encodings for the DMR (and the other response types as well) that a client might wish to receive. This is handled by having each normative responder hold a collection of alternative responders, each of which has its own suffix and regex. When *NormativeDMR* is built, it combines all of its alternative responder regex expression into a single combined regex. When *NormativeDMR* is asked if a request can be handled it simply evaluates the request URL path to see if it matches the combined regex. When the request is submitted for processing NormativeDMR locates the appropriate responder (using the individual regular expressions) and delegates the request to it.

Finally, the chosen responder will set the HTTP response headers appropriately (media types etc.) and then use the BesApi to build and issue a request to the target BES. This is done using a OPeNDAPClient object. The request is passed with the response output stream, and the OPeNDAPClient handles the transmission of the request and then the reception and retransmission of the BES response back to the client.

4.1.1. Service Conflation Issue

All this checking and "claiming" seems so involved. Yes. Yes it is. And it's partially a byproduct of the conflation of services. If I were to design it again I would follow the example set by the THREDDS

Data Server (TDS) and separate the services using a URL path prefix something like:

(Where ***path*** is a resource_id in the BES)

- *opendap/catalog/path* - Would produce a browser navigable HTML catalog of the servers holdings.
- *opendap/thredds/path* - Would produce a machine-readable/traversable THREDDS client catalog (XML) of the servers holdings.
- *opendap/dap4/path* - The DAP4 service and only the DAP4 service.
- *opendap/dap2/path* - The DAP2 service and only the DAP2 service.
- *opendap/file/path* - The file download service.
- etc.

I think each of these services might be implemented as a servlet. In Hyrax, all these services currently exist on the same URL path and a lot of the process described above is about the disambiguation of requests. I think this reorganization would reduce some of the code size and processing time, and potentially an ala-carte deployment where we can choose the services to include.

Chapter 5. Caching

The ***olfs*** utilizes a number of caches.

5.1. RequestCache

The RequestCache is used by every thread servicing a request. It's an object cache. It allows objects to be passed from on part the execution stack to another with it being carried as a parameter. Each thread that's servicing a request can safely put things it may need into the RequestCache. When the request/response is completed the RequestCache for the thread is cleared. (You can see this in the `opendap.coreServlet.DispatchServlet.doGet()` method)

The vast majority of the usages of RequestCache are responders caching a MediaType object as a set-up for request execution. This way if an error occurs the error handling apparatus can make good choices about how to encode the error response for the client.

5.2. BesNodeCache

To improve navigation performance, the ***olfs*** uses an in-memory LRU cache called BESNodeCache. This is used by the ***olfs*** to cache the BES responses to showNode commands (which is how the BES exposes it's dynamic navigable catalogs). These entries are held for a configurable period of time before a new request for the cached entry will cause the entry to be refreshed. This is a thread safe cache available to all processing threads.

5.3. BesSiteMap Cache

The BesSiteMap class is used to get and maintain a site map file for use with robots.txt The file is refreshed based on a configurable time period.

5.4. Thredds CatalogManager

Hard to know if this even gets used in the real world, so I'm going to skip longer explanation in favor of: For performance reasons the ***olfs*** can cache THREDDS catalogs in memory, and it is complicated when the catalogs need to be updated.

Chapter 6. Filters

- ***ClickJackFilter*** - Filter to stop the click jack hackery.
- ***BotFilter*** - Can be used to block access from single ip addresses, domains, or whatever your desires and regex acumen allow you to express. See ***olfs.xml*** for configuration details.
- ***IdFilter*** - Used for user authentication see Authentication/Authorization
- ***PEPFilter*** - Used for user authentication see Authentication/Authorization

Chapter 7. Resources

In the project directory you can find the resources included with the ***olfs*** when it's packaged for deployment. The default resources are located in *olfs/resources/hyrax* these include ***js***, ***jsp***, ***xml***, ***jnlp***, ***xslt***, and various ***xml*** files.

Some of the directories within resource may or may not be in use, ymmv.

7.1. olfs/resources/hyrax/`xsl`

A lot of the HTML produced by the server is built using the transforms found in this directory. At this moment (04/01/25) I know that all the files are hooked up to code in the server, but I suspect some of the things that use the ***xslt*** files may not be used themselves.

7.1.1. Version

- ***version.xsl*** - This is imported by nearly all the XSL transform docs, and is used to inject the Hyrax version string. When the builds are run, the key string `@HyraxVersion@` is replaced with the Version string value.

7.1.2. IFH Production

The IFH pages creation

- ***dap2ifh.xsl*** - Builds the DAP2 Data Request Form response from a DMR and passed parameters from the OLFS
- ***dap4ifh.xsl*** - Builds the DAP4 Data Request Form response from a DMR and passed parameters from the OLFS

7.1.3. Browsable Catalog UI

- ***dap4Contents.xsl*** - Builds a DAP4 ***contents.html*** page.
- ***node_contents.xsl*** - Builds a DAP2 ***contents.html*** page from a BES showNode response. (DirectoryDispatchHandler)

7.1.4. THREDDS Catalog production

- ***node_catalog.xsl*** - Builds THREDDS ***catalog.xml*** response from a BES showNode response. (ThreddsDispatchHandler)
- ***besNodeToDatasetScanCatalog.xsl*** - The THREDDS Things, ***3T***.
- ***threddsCatalogIngest.xsl - 3T***
- ***threddsCatalogPresentation.xsl - 3T***
- ***threddsDatasetDetail.xsl - 3T***
- ***threddsMetadataDetail.xsl - 3T***

7.1.5. RDF Responses

- ***anyXml2Rdf.xsl*** - Used to produce both the DAP2 and DAP4 RDF responses for the server. Oddly this seems to be a pointless empty wrapper for ***xml2rdf.xsl***
- ***xml2rdf.xsl*** - Simply included in ***anyXml2Rdf.xml*** `<xsl:include href="xml2rdf.xsl"/>`

7.1.6. w10n

- ***showNodeToW10nCatalog.xsl*** - Makes w10n collection pages from a BES showNode response.
- ***w10nDataset.xsl*** - makes w10n dataset pages from a DAP3.2 DDX from the BES.

7.1.7. Viewers

- ***webStartDataset.xsl*** - Used by the viewers servlet produce an available viewers page for a dataset.

7.1.8. Obscure DAP4 Things

- ***asyncResponse.xsl*** - Used by a prototype DAP4 asynchronous response implemented in dap4Responders.DataResponse.NormativeDR.handleStoreResultResponse(). *I bet this feature is never used.*
- ***datasetServices.xsl*** - Used to create the DAP4 Dataset Services Response page. *Probably not used.*

7.2. olfs/resources/hyrax/`ngap`

The *olfs/resources/hyrax/ngap* directory contains the resources that we use amend/replace things from the regular build to produce our NGAP builds. Things like a custom landing page and the libraries to let us use ***memcached*** to manage session state in our multi-host deployments.

Chapter 8. Configuration

The ***olfs*** utilizes several configuration files. The primary one, *olfs.xml*, controls ***olfs*** behavior and defines the locations of one or more BES services that are needed for the Hyrax to function.

8.1. Locating the configuration at startup.

When the server starts the code in `opendap.coreServlet.DispathServlet.init()` utilizes the ServletUtil class methods to locate the configuration. The `ServletUtil.getConfigPath()` method checks the following locations:

1. */etc/olfs*
2. */usr/share/olfs*
3. *$CATALINA_HOME/webapps/* deployment_context */WEB-INF/conf*

If neither of the first two locations are located then the default configuration bundled in the war file will be used.

If either of the first two locations is found, and if the configuration is not in that location, and the running service has write permission to the identified location, then the service will copy its default configuration to the identified location.This creates a configuration that can be easily modified by the operator and that will persist across updates and/or redeployments of the ***olfs***.

The *DispatchServlet.init()* method also loads a vector with new instances of its DispatchHandler objects that it will utilize to service requests. More on this below.

8.2. olfs.xml

This file contains the core configuration for the ***olfs***. It defines a number of service behaviors and also the location of the BES(s) to be used when handling requests.

8.3. user-access.xml

This file is an authentication and permissions control file that allows the operator to deploy a service that can make use of OAUth2 Single Sign On (SSO) to authenticate users. The file also defines the various operational permissions the authenticated users will have (via groups and roles)

8.4. logback.xml and logback-test.xml

These files provide the logging framework behaviors: Log file names, log file content, and what things will be logged at each logging level.

8.5. catalog.xml

The top level static THREDDS client catalog for the machine traversable THREDDS service. This file nominally contains a catalog ref to the dynamically generated Hyrax THREDDS catalogs:

```
<catalogRef xlink:href="/opendap/hyrax/catalog.xml" xlink:title="Hyrax Catalog"
name="Hyrax Catalog" />
```

It can be expanded to create a much more extensive static catalog by creating a filesystem tree of catalog files linked by the appropriate catalogRef elements.

8.6. viewers.xml

This provides the configuration for the various Java Web Start applications (IDV Viewer, NetcdfToolsUI, etc.) and WebServiceHandlers (NcWMS, Godiva)

8.7. webstart.xml

I think that this can be retired, but that assertion needs to be tested.

Chapter 9. Servlets

The servlet implementations subclass HttpServlet. The servlets implement the various services that the ***olfs*** supports. Some of these rely on one or more *DispatchHandler* implementations to perform the heavy lifting of answering the requests. There is not a one to one and onto relationship with things I think of as a service and the servlets. They all implement a service, but some implement several services.

- Package: **opendap.aggregation**
    - `AggregationServlet.java` - Implements the Hyrax Aggregation Service API
- Package: **opendap.auth**
    - `PDPService.java` - This standalone service is an implementation of a Policy Decision Point which can be used as part of an authentication and access permissions activity. I can be run on a remote system and asked if a particular user/group/role is allowed access to a particular resource.
- Package: **opendap.bes**
    - `BESSiteMapService.java` - This service uses the BES to construct and cache a Site Map of the data holdings in a particular deployment of Hyrax.
- Package: **opendap.build_dmrpp**
    - `BuildDmrppServlet.java` - This service can be used to have Hyrax create and return a dmr++ document for a particular dataset. *It seems to be the case that this service has been "overwhelmed by events" (OBE) and might be retired.*
- Package: **opendap.coreServlet**
    - `DispatchServlet.java` - This service is the primary Hyrax service and uses a number of DispatchHandler implementations to provide DAP, file access, and other services.
    - `DocServlet.java` - This service provides client access to the static documents held by the ***olfs*** such as image, css, and javascript files.
- Package: **opendap.gateway**
    - `DispatchServlet.java` - Implements the Hyrax Gateway Service. *We should probably review this with an eye towards its retirement.*
- **Package: opendap.viewers**
    - `ViewersServlet.java` - This servlet is used create JavaWebStart documents that can can be utilized by a client to launch/access external applications such as *IdvViewer*, *NetCdfTools*, and *AutoPlot*. It can also direct users to externally running web services such as *NcWms* and/or *Godiva*.
- Package: **opendap.w10n**
    - `W10nServlet.java` - Implements the w10n data access api for the Hyrax data holdings.
- Package: **opendap.wcs.v2_0.http** - *There are problems with WCS in general, and it would be worth evaluating the retirement potential for this package.*
    - `Servlet.java` - Provides a WCS 2.0 implementation. This works, but requires the operator to

perform configuration work for each "collection" in order to identify the domain and range variables for the collections datasets.

Chapter 10. Dispatch Handlers

These classes implement the `opendap.coreSevlet.DispatchHandler` interface and are used by the `CoreServlet` to direct the client requests to the appropriate software for generation and transmission of the response.

- Package: **opendap.bes**
  - `BesDapDispatcher.java` - This *DispatchHandler* implementation provides the DAP2 and DAP4 services for Hyrax, which includes data access services and the production of the HTML Data Request Forms. It does this by utilizing a collection of *Dap4Responder* implementations. Each of which handles a very specific task, typically commanding the BES to produce a particular response.
  - `BESThreddsDispatchHandler.java` - This DispatchHandler uses a combination of the BES `showNode` interface, and a XSL transform file along with state information to generate THREDDS client catalog responses and return them to the user.
  - `DirectoryDispatchHandler.java` - This DispatchHandler uses the BES `showNode` interface, and a XSL transform file to produce the browser navigable "blue-bar" pages that express the data holdings of the service as a directed graph.
  - `FileDispatchHandler.java` - Used to transmit files from the BES to the user. If the source file is seen as a data file byt the BES this will be blocked unless specifically enabled in the *olfs.xml* configuration file by uncommenting the `<AllowDirectDataSourceAccess />` element.
  - `VersionDispatchHandler.java` - This combines the BES version information and ***olfs*** version information to make and return a combined XML version document to the requestor.
- Package: **opendap.build_dmrpp**
  - `BuildDmrppDispatchHandler.java` -
- Package: **opendap.coreServlet**
  - `NoPostHandler.java` - This is used when POST request submissions are disabled, which is determined by the presence of the `<HttpPost …/>` element in *olfs.xml* file.
- Package: **opendap.gateway**
  - `DispatchHandler.java` - This handler implements the Gateway Service. It is one of the DispatchHandlers loaded by the opendap.coreServlet.DispatchServlet
- Package: **opendap.nciso**
  - `IsoDispatchHandler.java`- Deprecated (should be moved to *retired*)
  - `RubricDispatchHandler.java` - Deprecated (should be moved to *retired*)
- Package: **opendap.ncml** - *I think this whole package might be retired.*
  - `NcmlDatasetDispatcher.java`
- Package: **opendap.ngap**
  - `NgapDispatchHandler.java` - This is the NGAP handler, a child od BEsDapDispatcher. It claims any request that begins with its prefix string (default: *ngap/*) and then uses it's parent class and a specialization of the BesApi, NgapBesApi, to build BES commands to service the

request.

- Package: **opendap.threddsHandler**
    - `StaticCatalogDispatch.java` - The ***olfs*** server static THREDDS client catalogs, and this is the class that does this. Static catalogs exist on the host filesystem and are integrated with the dynamic catalogs built by interacting with the BES.
- Package: **opendap.wcs.v2_0.http** - *I think we should retire this package.*
    - `FormHandler.java`
    - `HttpGetHandler.java`
    - `SoapHandler.java`
    - `XmlRequestHandler.java`

Chapter 11. DapResponders

The abstract class Dap4Responder class embodies a number of general operations needed for:

- Identifying requests that should be handled by an instance of the class.
- HTTP client/server content negotiation for DAP4 (DAP2 doesn't support this)

This could use a refactor to condense the hierarchy of inheritance from 4 classes to 2.)

11.1. DAP2 Responders

Responders that handle all the DAP2 things. All are children of Dap4Responder. Yeah. I know. Go figure.

- Package: **opendap.bes.dap2Responders**
    - `Ascii.java` - Uses the BES to produce and transmit the DAP2 ASCII encoded data response.
    - `CovJson.java` - Uses the BES to produce and transmit the DAP2 Coverage JSON encoded data response.
    - `CsvData.java` - Uses the BES to produce and transmit the DAP2 CSV encoded data response (which is fundamentally the same as the DAP2 ASCII response).
    - `Dap2Data.java` - Uses the BES to produce and transmit the DAP2 data response.
    - `Dap2IFH.java` - Uses the BES to retrieve the DAP3.2 DDX XML document. This is fed, along with injected state information, into an XSL transform to produce the DAP2 Data Request Form (aka the Interface From Hell, aka the IFH) and transmit form page to the requesting client.
    - `DAS.java` - Uses the BES to produce and transmit the DAP2 DAS response.
    - `DatasetInfoHtmlPage.java` - Uses the BES to produce and transmit the DAP2 Dataset Info Page response.
    - `DDS.java` - Uses the BES to produce and transmit the DAP2 DDS response.
    - `DDX.java` - Uses the BES to produce and transmit the DAP3.2 DDX response (unique to Hyrax, not part of the DAP2 specification, a stepping stone to DAP4).
    - `GeoTiff.java` - Uses the BES to produce a DAP2 data response and encode it as a GeoTiff file.
    - `GmlJpeg2000.java` - Uses the BES to produce a DAP2 data response and encode it as a GMLJpeg2000 file.
    - `Ijson.java` - Uses the BES to produce a DAP2 data response and encode it as an "instance" json (.ijsn) response.
    - `Iso19115.java` - Uses the BES to produce a DAP3.2 DDX response and then applies an XSL transform to produce ISO-19115 metadata document.
    - `Iso19115Rubric.java` - Uses the BES to produce a DAP3.2 DDX response and then applies an XSL transform to produce an HTML page the shows how the metadata does and does not conform to the ISO-19115 expectations.

    - `Json.java` - Uses the BES to produce a DAP2 data response and encode it as a json (.json) response.
    - `Netcdf3.java` - Uses the BES to produce a DAP2 data response and encode it as a netcdf-3 file.
    - `Netcdf4.java` - Uses the BES to produce a DAP2 data response and encode it as a netcdf-4 file.
    - `RDF.java` - Uses the BES to produce a DAP3.2 DDX response and then applies an XSL transform to an HTML page to represent the document as RDF.
    - `XmlData.java` - Uses the BES to produce a DAP2 data response and encode it as an XML document.

11.2. Generic Responders

- Package: **opendap.bes.dap4Responders**
    - `FileAccess.java` - Used to transmit files from the BES to the requesting client.
    - `Version.java` - Builds and returns the Hyrax combined version response document (XML).

11.3. DAP4 Responders

Responders that handle all the DAP4 things.

- Package: **opendap.bes.dap4Responders.DataResponse** This package contains responders that return data in various encodings.
    - `CovJsonDR.java` - Uses the BES to produce and transmit the DAP4 data response encoded as Coverage JSON (may not have a companion implementation in the BES)
    - `CsvDR.java` - Uses the BES to produce and transmit the DAP4 data response encoded as CSV.
    - `GeoTiffDR.java` - Uses the BES to produce and transmit the DAP4 data response encoded as a GeoTiff file.
    - `GmlJpeg2000DR.java` - Uses the BES to produce and transmit the DAP4 data response encoded as a GML JPEG 2000 file.
    - `IjsonDR.java` - Uses the BES to produce and transmit the DAP4 data response encoded as instance JSON (.ijsn)".
    - `JsonDR.java` - Uses the BES to produce and transmit the DAP4 data response encoded as JSON.
    - `Netcdf3DR.java` - Uses the BES to produce and transmit the DAP4 data response encoded as a netcdf-3 file. Note that the netcdf-3 data model is a subset of the DAP4 data model and not all DAP4 data content m,ay encoded as netcdf-3.
    - `Netcdf4DR.java` - Uses the BES to produce and transmit the DAP4 data response encoded as a netcdf-4 file.
    - `NormativeDR.java` - Uses the BES to produce and transmit the normative DAP4 data response.
    - `XmlDR.java` - Uses the BES to produce and transmit the DAP4 data response encoded as an XML data response.
- Package: **opendap.bes.dap4Responders.DatasetMetadata** This package contains responders

that return metadata in various encodings.

- `HtmlDMR.java` - Uses the BES to produce and return the DMR response which is used, along with an XSL transform and several state variable to produce the DAP4 Data Request Form.
- `IjsonDMR.java` - Uses the BES to produce and transmit the DAP4 metadata response encoded as an instance JSON (.ijsn).
- `JsonDMR.java` - Uses the BES to produce and transmit the DAP4 metadata response encoded as JSON.
- `NormativeDMR.java` - Uses the BES to produce and transmit the normative DAP4 metadata response (XML).
- `RdfDMR.java` - Uses the BES to produce and transmit the DAP4 metadata response encoded as RDF.
- `XmlDMR.java` - Uses the BES to produce and transmit the DAP4 metadata response encoded as XML.

* Package: **opendap.bes.dap4Responders.DatasetServices** This package contains responders that return the DAP4 Dataset Service Response (DSR) in various encodings. *I think at this opoint this package is obviated as the DSR while defined in the DAP4 specification does not enjoy any know useful implementation.*
   - `HtmlDSR.java` - Uses the BES to retrieve the DSR response and return it encoded as HTML by using a XSL transform.
   - `NormativeDSR.java` - Uses the BES to produce and transmit the DSR response (XML).
   - `XmlDSR.java` - Uses the BES to produce and transmit the DSR response (XML).
* Package: **opendap.bes.dap4Responders.Iso19115**
   - `IsoDMR.java`- Uses the BES to produce a DAP4 DMR response and then applies an XSL transform to produce ISO-19115 metadata document.
   - `IsoRubricDMR.java` - Uses the BES to produce a DAP4 DMR response and then applies an XSL transform to produce an HTML page the shows how the metadata does and does not conform to the ISO-19115 expectations.

Chapter 12. Authentication/Authorization

For NASA deployments OLFS can be configured to provide authentication services using an (almost) OAuth2 implementation in the ***opendap.auth*** package. The implementation follows the Identity Provider (IdP) plus Policy Decision Point (PDP) pattern.

12.1. Controls

You must first build an authentication/authorization enabled the web archive file, and then you will need to configure the deployment.

12.1.1. Enable Authentication/Authorization

To enable authentication: Edit the file ***resources/hyrax/WEB-INF/web.xml*** to enable to two authentication filters ***IdFilter*** and ***PEPFilter***. Then the service must be rebuilt and a new WAR file generated with this change in place.

12.1.2. Configure Authentication/Authorization

All the configuration happens in the ***user-access.xml*** file located in the configuration directory.

12.2. Authentication - aka the Identity Provider (IdP)

Authentication is the operation wherein a users software client establishes that it is working on behalf of a particular user. Use the abstract class ***IdProvider*** to capture the common API for an identity provider and provide two implementations, ***UrsIdP*** and ***TomcatRealmIdP***. In theory the ***TomcatRealmIdP*** should work the Tomcat Memory Authentication Realm, but this has not been verified in years. The ***UrsIdP*** knows how to authenticate user with NASA’s Earth Data Login (EDL) service. In NASA speak, ***UrsIdP*** is an *EDL client*. The IdFilter provides the core of the authentication functionality and handles saving request information (Like target URL and query string) in a user session so that it can be used to redirect the client to their original request target when the client returns from a ***Single Sign On*** authentication journey.

12.2.1. EDL Tokens

The IdFilter will also accept an HTTP ***Authorization*** headers with a token.

Typically, ***Authorization*** headers look like:

```
Authorization: Bearer A_LONG_STRING_OF_TOKEN_THINGS
```

These can be submitted to the OLFS with the request and no trip need be made to EDL by the client. When using authorization tokens, sessions do not need to be maintained by the client, although doing so makes the transaction faster still.

12.2.2. Classes

The IdP is composed of:

- ***IdProvider*** - Theis abstract classes contains common machinery/API is used to parent specific implementations of a provider.
- ***UrsIdp*** - Implements an IdProvider that utilizes Earth Data Login (EDL, aka URS) to authenticate a user. All the URS action is in here!
- ***TomcatRealmIdp*** - In theory this IdProvider implementation works with Tomcat Realms (like the MemoryRealm). It’s been a long time since it was tested.
- ***IdManager*** - This **singleton** class holds a collection of IdProviders, but in practice only ever has one. If NASA follows through with allowing user to choose between EDL Google authentication then we’ll be right in here making this work!
- ***IdFilter*** - This class does the heavy lifting of authentication

12.3. Authorization - aka the Policy Decision Point (PDP)

In this software we define users, groups, and roles in order to create role based access policies. In ***user-access.xml*** configuration file we define a group based on one or more "user" definitions:

```
<!-- Define the group urs_users as all users in the authContext urs -->
<group id="urs_users" >
    <user idPattern="^(?!\s*$).+" authContext="urs" />
</group>
```

A role is defined as one more groups and each policy can be applied to one or more roles.

```
<!-- The only group in the role "users" is the "urs_users" group. -->
<role id="users">
    <group id="urs_users" />
</role>
```

Here’s a policy that allows anyone to access the image and navigable catalog files in Hyrax.

```
<Policy class="opendap.auth.RegexPolicy">
    <role>.*</role>

<resource>^.*((\/docs\/.*)|(\/)|(\.(css|png|jpg|ico|gif|xsl|jsp|svg))|(\/contents\.htm
l)|(\/catalog\.html)|(\/catalog\.xml)|(logout)|(login)|(\/version))$</resource>
    <queryString>.*$</queryString>
    <allowedAction>HEAD</allowedAction>
    <allowedAction>GET</allowedAction>
</Policy>
```

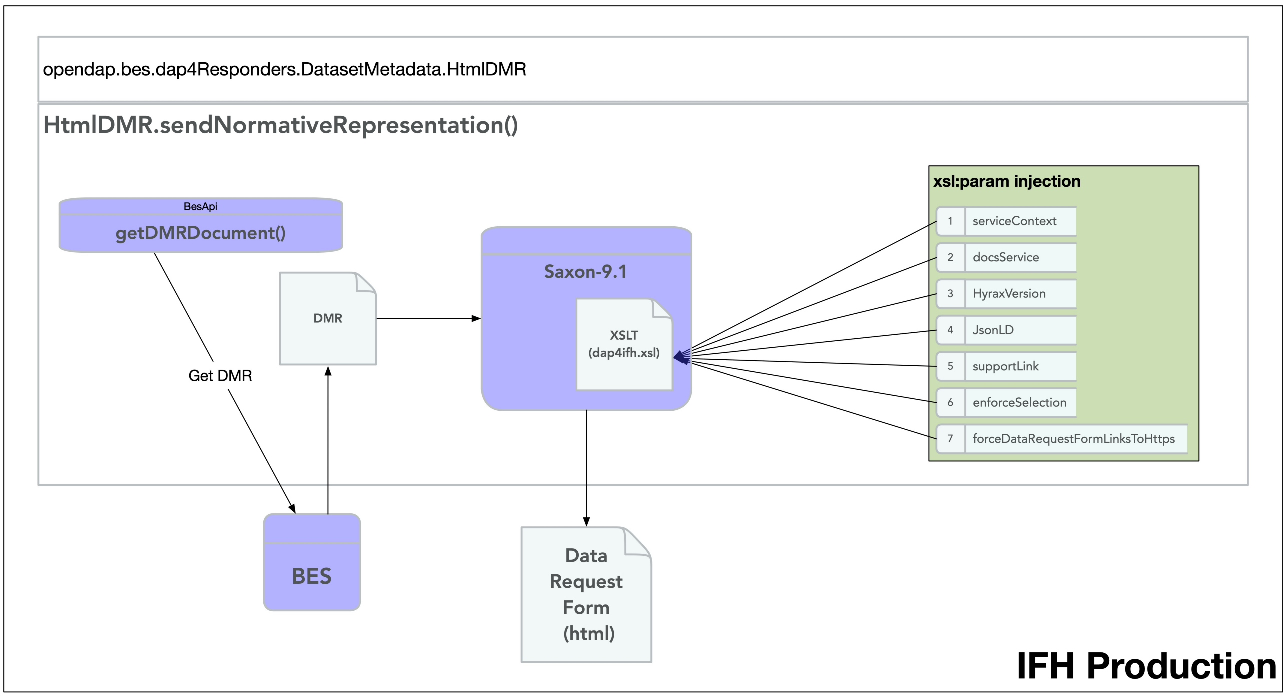
This allows non-authenticated users to look around , and when something bad happens they get a web page rendered with css and images, which they wouldn’t if this rule is not included.

12.3.1. Classes

- ***Policy*** - This interface defines the policy API
- ***RegexPolicy*** - A policy the uses Regx matching on the URL path and the query string to evaluate access. If the request matches the regex patterns users/groups/roles held by the Policy are allowed access.
- ***PolicyDecisionPoint*** - This abstract class is the parent of ***RemotePDP*** and ***SimplePDP***.
- ***PEPFilter*** - This Filter is an embodiment of a policy enforcement point. It holds an instance of PolicyDecisionPoint which is used to evaluate incoming requests to determine if access will be allowed. If the request is not permitted, and they aren’t logged in, then they are redirected to the login endpoint (aka IdFilter) to get that sorted.
- ***PDPService*** - This child of HttpServlet can be runs as a standalone PDP service, accepting evaluation requests and responding with simple permitted or not permitted.
- ***RemotePDP*** - Defines a policy decision point somewhere on the internet. With a remote PDP the server sends the uid, authContext, url path, query , and http verb (GET, POST, etc.) and the remote PDP responds with permitted or not permitted.
- ***SimplePDP*** - *"Simple is often better"* - Ned Greene 1984 This is the policy decision point that we use. It is locally hostsed and configured.

Chapter 13. The IFH

Here's a simple picture that shows how the DAP4 IFH is produced. The DAP2 IFH production process is nearly identical, but deprecated.



What this picture does not capture is the hot mess inside the ***dap4ifh.xsl*** operation. The XSL takes the DMR and builds a javascript program that enables the form page to operate, and then it wraps an HTML page with controls for selection and subsetting around the dynamically generated javascript code. One of the tasks is to create javascript variable names for the things in the DMR that can be selected and, if possible, subset. Currently, this includes dap4 Variables and Groups. But hopefully soon will include dap4 Dimensions.

There are known issues with the current approach. One such issue: The XSL seeks to use the actual variable names in the DMR as the javascript variable names, and that's a problem because the allowed characters in javascript variable names encompass just a small subset of the allowed characters in a DAP4 Variable | Attribute | Dimension name. For example, a variable name containing a space character is allowable in DAP4, but it will break this code because the space isn't allowed in javascript variable names. This is a legacy approach that needs to be refactored so that the DMR variable names are available for display purposes, but under the hood the associated javascript variable names are valid. Personally I would like to keep some form of the original DMR name just because reading the resulting code in the Form response is hard enough without anonymizing the variable names!

13.1. Constraint Issues

There are problems with the way the form builds dap4 constraint expressions.

- **Special characters** - When characters like the space character occur in a variable name, that name must be surrounded by double quotes in the dap4 constraint for the Hyrax constraint expression parser to handle it correctly. The form needs to detect these characters and double quote these names as needed. What’s that? A list? You want List?

  - This double quoting business is not mentioned in the DAP4 specifcation. Seems like we need to evaluate our CE parser to see if it really has to have the quotes? The names in used in the *dap4.ce* are FQNs after all, and if the name is properly escaped then it seems to me that the parser should be able to handle it. (see the table at the end of the Fully Qualified Names section of the DAP4 specification)

- **Dimension based subsetting** - The IFH currently requires that the user provide a subsetting expression for every variable, even if the variables have common shared Dimensions. The form should allow subsetting by Dimension: The user creates a subset expression for just the Dimensions and then all the selected variables are subset based on the Dimension subsets. This would be and advanced IFH assignment.

- Subset expressions can be simplified - If the user selects an array and the subset expression in the form is asking for every element, then the array index subset expression should be elided from the constraint expression.

Chapter 14. Testing

- Virtually no unit tests exist for the ***olfs***.
- There are regression tests. The tests exist in their own GitHub repo, the hyrax_regression_tests project. They require a fully operational Hyrax in order to be run.