OPeNDAP Aggregation Server Guide
Version 1.1

John Caron
Tom Sgouros

December 1, 2002

Preface

This document describes version 0.6 of the OPeNDAP Catalog/Aggregation Server, a service designed to create logical groupings of individual data files. The individual files may be local netCDF files, or remote files, served from an OPeNDAP server. Using the Aggregation Server, you can effectively merge these disparate files into what seems to be one larger file, which may be sampled using the standard OPeNDAP constraint expressions. The files need not be local files, nor need they be on the same remote server.

The Aggregation Server creates logical groupings of OPeNDAP data files.

You can also use the Aggregation Server to serve individual netCDF files.

This document assumes you are familiar with the operation of OPeNDAP servers and software. If this is not the case, you may want to consult *The DODS User Guide* or *The DODS Quick Start Guide* first. You will also need a basic understanding of the syntax and construction of XML documents.

Contents

List of Figures

1
Theory

The OPeNDAP Catalog/Aggregation Server is an OPeNDAP server that creates virtual datasets from collections of other OPeNDAP datasets or from NetCDF files. It can also serve single NetCDF files as (non-aggregated) OPeNDAP datasets. It uses Thematic Realtime Environmental Data Distributed Services (THREDDS) catalogs to specify what datasets it serves, and how to aggregate them. The specification is formatted as an XML document.

OPeNDAP data is accessed by files, but that's not always the way users want it.

The Open Source Project for Network Data Access Protocol (OPeNDAP) is organized around the concept that the relevant unit of data storage is a disk file. In order to get or retrieve data, you must know what file the desired data is in. To allow users to identify specific files, data providers create lists of files, or catalogs, that may be browsed or searched. This approach makes certain aspects of data management simpler, and is a natural consequence of several popular data storage APIs which work the same way. However, many users may perceive it as a burden to have to keep track of file names, which are not always chosen to be easy to remember. What's worse, it becomes quite awkward if the slice of data you happen to be interested in spans many files. Looking for a time series in an archive of satellite data, for example, might require thousands of requests to individual data files.

The OPeNDAP Catalog/Aggregation Server is designed to accommodate these problems, providing a single point of entry to collections of many files. Conceptually, it consists of two components: a catalog of data files, and the smarts necessary to determine how to use those data files to satisfy user requests. The files in question may be either local files, existing on the same computer as the server, or they may be remote files, specified with an OPeNDAP URL. The catalog is a file of XML declarations which identify how individual data files can be aggregated to look like single larger files.

To the end-user, aggregated data looks the same as other data.

In operation, the Aggregation Server accepts a query from a user, and determines how to use the data files listed in its catalog to fulfill that query. After making this determination, the Aggregation Server makes the subsidiary queries necessary to satisfy the original request, aggregates this data into the form expected by the user who initiated the request, and returns the result to that user. The user need not even know that the result is an aggregation. Though a complex aggregation will obviously take longer than a simple OPeNDAP data request, there should be no other differences in the user interaction.

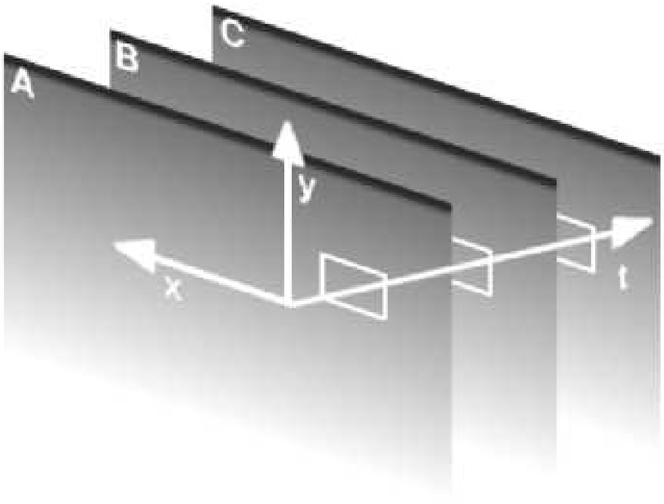

Figure 1.1: JoinNew aggregation

1.1 Methods of Aggregation

Aggregated data is data that should belong together.

Any data can be included in a given data catalog, but not all data can be (or should be) aggregated. Two data files that can (usefully) appear joined together in an Aggregation Server might be records of the same kinds of data on different dates, or different kinds of data at the same location, or data at adjoining locations at the same time. That is, the data to be aggregated must share some common features to be worth the trouble to aggregate them. If you can't imagine a user wanting to make the same query to two different data files, they probably aren't worth aggregating.

Even though two data files are not aggregated, they may still be included as part of the same data catalog. The catalog can help users find data, even though it isn't aggregated.

The OPeNDAP Catalog/Aggregation Server can handle three different forms of aggregation. We call these three *JoinNew*, *JoinExisting*, and *Union*. We'll examine each of these in turn.

> **NOTE:** As of version 0.6 of the Aggregation Server, only Grid and Array data types can be aggregated.

JoinNew The *JoinNew* form of aggregation is used to join datasets along a new dimension. For example, if you have a set of measurements taken of the same spatial area at different dates, you could arrange these in order by time to create a time series of measurements. If a user wanted to see the time evolution of a measurement at a subsection of the larger measured area, the situation might look sort of like what's pictured in figure 1.1.

JoinNew is used to aggregate data along a data type not in any file.

Here, A, B, and C represent sets of measurements in the X and Y dimensions. The three sets are aggregated along the Z axis. This means that none of A, B, or C have a Z variable in those sets, but they all contain (or represent) a single Z value. In our example above, X and Y are spatial dimensions, and Z would be time.

JoinExisting If you have several datasets that consist of data in adjoining regions of space or time, you may be able to aggregate them with the *JoinExisting* form of aggregation. For example, if you have a time series that begins right after another one ends, you can combine these two along the time axis. Similarly, if you have two spatial grids, whose edges adjoin, you might be able to join them along the dimension orthogonal to the common edge. Something of this nature is shown in figure 1.2, where three datasets, each of which contain independent variables X, Y, and Z, are joined on the Z axis.

JoinExisting is used to combine two adjoining datasets.

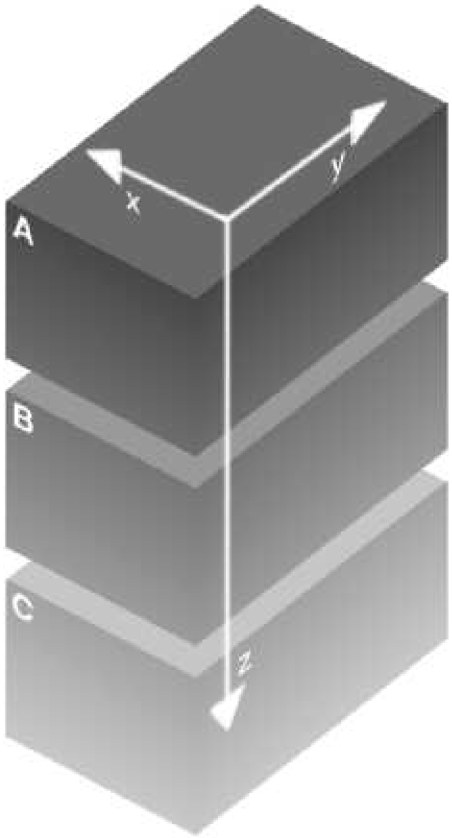

Figure 1.2: JoinExisting aggregation

Union is used to combine datasets consisting of different data types.

Union If you need to aggregate datasets that cover the same space and time areas, but consist of different data types, you can use the *Union* aggregation to join them. For example, you might have a grid of satellite sea surface temperature values, and another grid of wind speed observations. You can use the *Union* aggregation to combine the two into one dataset containing both temperature and wind speed.

Now that you have the idea of what is meant by aggregation, the next section will show how to specify the method and parameters for aggegration using the necessary XML syntax. The following chapter will explain how to install the server. Chapter 3 describes how to configure the aggregation server so it will show your data in the way you want.

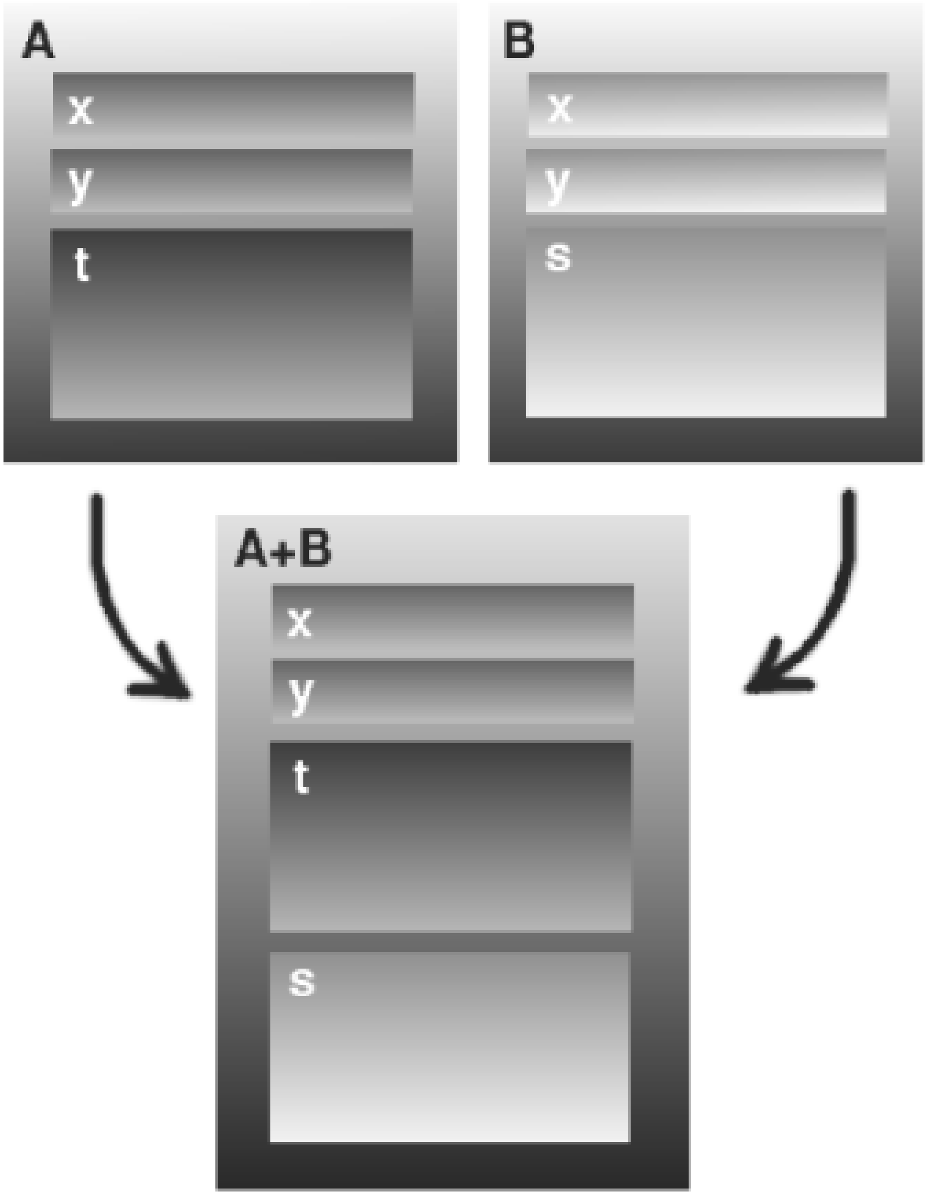

Figure 1.3: Union aggregation

1.2 A little more detail

Here are some worked-out examples of aggregation types. If you don't know what the elements of an OPeNDAP Data Descriptor Structure (DDS) mean, or aren't familiar with the basic OPeNDAP data types, please refer to *The DODS User Guide*.

1.2.1 JoinNew Aggregation

The *JoinNew* aggregation type joins variables along a new dimension. The dimension and a coordinate variable are created and values for the coordinates are specified in the *aggregation* element:

```
<aggregation serviceName="ISCCP" aggType="JoinNew"
             varName="time">
  <fileAccess urlPath="cldfrc/isccp.8501.bin" coord="Jan"/>
  <fileAccess urlPath="cldfrc/isccp.8502.bin" coord="Feb"/>
  <variable name="cldfrc"/>
</aggregation>
```

This XML fragment will combine these two datasets:

```
Dataset {
  Float32 cldfrc[lat = 180][lon = 360];
} isccp_c2;

Dataset {
  Float32 cldfrc[lat = 180][lon = 360];
} isccp_c2;
```

into this one:

```
Dataset {
  String time[time = 2];
  Float32 cldfrc[time = 2][lat = 180][lon = 360];
} ISCCP/cldfrc;
```

The coordinate variable `time` has been assigned the values "Jan" for dataset 8501 and "Feb" for dataset 8502, as specified in the *aggregation* tag.

> **NOTE:** *JoinNew* aggregations will automatically join all variables of type *Grid*. Variables of type *Array* may also be joined, but must be explicitly listed in a *variable* element. In the example here, the *variable* element specified that all arrays called `cldfrc` were to be joined.

There are several options for assigning values to the coordinate variable of the joined dimension of *JoinNew* aggregations; see a description of the *varType* and *varUnit* attributes of the *aggregation* element. The most common use of *JoinNew* aggregations is to add a time dimension. The *dateFormat* attribute is used to convert date/time strings into other units. In the following example:

```
<aggregation aggType="JoinNew" varName="time" varType="int"
    varUnit="days since 0000-01-01 00:00:00"
    dateFormat="yy/M/d:HH:mm:ss z">
<fileAccess
    urlPath="20020101.dat" coord="2002/1/1:00:00:00 GMT"/>
<fileAccess
    urlPath="20020102.dat" coord="2002/1/2:00:00:00 GMT"/>
<fileAccess
    urlPath="20020103.dat" coord="2002/1/3:00:00:00 GMT"/>
</aggregation>
```

The time data strings are read in using the *SimpleDateFormat* string `yyyy/M/d:HH:mm:ss z`, then converted to the unit specified with the *varUnit* attribute:

```
days since 0000-01-01 00:00:00
```

using the *ucar.units* package and stored in the dataset as an `int32`, so the time coordinate values will look like:

```
time[3]
0, 1, 2
```

1.2.2 JoinExisting Aggregation

The *JoinExisting* aggregation type joins variables along an existing dimension. The dimension must have a coordinate variable and is named in the *aggregation* element:

```
<aggregation serviceName="local" varName="time"
    aggType="JoinExisting">
  <fileAccess urlPath="cdc/air.1948.nc"/>
  <fileAccess urlPath="cdc/air.1949.nc"/>
</aggregation>
```

The above aggregation declaration joins these two datasets:

```
Dataset {
  Float32 level[level = 17];
  Float32 lat[lat = 73];
  Float32 lon[lon = 144];
  Float64 time[time = 366];
  Grid {
    ARRAY:
      Int16 air[time = 366][level = 17][lat = 73][lon = 144];
    MAPS:
      Float64 time[time = 366];
      Float32 level[level = 17];
      Float32 lat[lat = 73];
      Float32 lon[lon = 144];
  } air;
} cdc/air.1948.nc;

Dataset {
  Float32 level[level = 17];
  Float32 lat[lat = 73];
  Float32 lon[lon = 144];
  Float64 time[time = 365];
  Grid {
    ARRAY:
      Int16 air[time = 365][level = 17][lat = 73][lon = 144];
    MAPS:
      Float64 time[time = 365];
      Float32 level[level = 17];
      Float32 lat[lat = 73];
      Float32 lon[lon = 144];
  } air;
} cdc/air.1949.nc;
```

The resulting DDS is shown below in Example 1 on page 15. Notice the `time` dimension of the `air` grid. In this case, all variables of type *Array* or *Grid* with the specified outermost dimension are joined into a single variable in the aggregated dataset. All other non-joined variables, and all attributes are taken from the first of the joined datasets.

Example 1

```
Dataset {
  Float32 level[level = 17];
  Float32 lat[lat = 73];
  Float32 lon[lon = 144];
  Float64 time[time = 731];
  Grid {
    ARRAY:
      Int16 air[time = 731][level = 17][lat = 73][lon = 144];
    MAPS:
      Float64 time[time = 731];
      Float32 level[level = 17];
      Float32 lat[lat = 73];
      Float32 lon[lon = 144];
  } air;
} local/MeanAir;
```

1.2.3 Union Aggregation

A *Union* aggregation type creates the union of all the variables in the component datasets. Assume we have two datasets, with the following two DDS's:

```
Dataset {
  Float32 lat[lat = 21];
  Float32 lon[lon = 360];
  Grid {
    ARRAY:
      Int16 cldc[lat = 21][lon = 360];
    MAPS:
      Float32 lat[lat = 21];
      Float32 lon[lon = 360];
  } cldc;
} coads/cldc.mean.nc;

Dataset {
  Float32 lat[lat = 21];
  Float32 lon[lon = 360];
  Grid {
    ARRAY:
      Int16 lflx[lat = 21][lon = 360];
    MAPS:
      Float32 lat[lat = 21];
      Float32 lon[lon = 360];
  } lflx;
} coads/lflx.mean.nc;
```

Using the following XML fragment to specify the aggregation creates a DDS like the one in Example 2 on page 16.

```
<aggregation serviceName="COADS" aggType="Union">
 <fileAccess urlPath="cldc.mean.nc"/>
 <fileAccess urlPath="lflx.mean.nc"/>
</aggregation>
```

The resulting DDS (and the accompanying Data Attribute Structure (DAS)) is the union of all the individual files' DDS. The first time a variable is encountered it is added to the combined DDS, subsequent variables with the same name are ignored. In the above example, the top level variables `lat` and `lon` are therefore taken from the `cldc.mean.nc` dataset. See figure 1.3.

Example 2

```
Dataset {
  Float32 lat[lat = 21];
  Float32 lon[lon = 360];
  Grid {
    ARRAY:
      Int16 cldc[lat = 21][lon = 360];
    MAPS:
      Float32 lat[lat = 21];
      Float32 lon[lon = 360];
   } cldc;
  Grid {
    ARRAY:
      Int16 lflx[lat = 21][lon = 360];
    MAPS:
      Float32 lat[lat = 21];
      Float32 lon[lon = 360];
  } lflx;
} local/coads;
```

2

Practice I: Installing the OPeNDAP Catalog/Aggregation Server

The Aggregation Server is a Java servlet that implements an OPeNDAP server both for netCDF files and for “aggregation” datasets. Version 0.6 and after requires Java version 1.4.

2.1 Install Jakarta-Tomcat Server

To begin installation, you must install the Tomcat servlet engine. The Aggregation Server will work under either Tomcat 3.3 or 4.0. Install Tomcat in standalone mode. This is the default configuration, using port 8080 and the default tomcat.conf and server.xml files.

The Aggregation Server requires the Tomcat servlet engine.

Here is an example installation, assuming that the Aggregation Server name is dodsC, so that all URLS start with *server*/`dodsC`, where *server* is whatever the name of your machine is.

❶ Edit `$TOMCAT_HOME/bin/startup.sh` and make sure it contains the equivalent of the following:

```
export TOMCAT_HOME=/usr/local/jakarta-tomcat
export JAVA_HOME=/usr/local/jdk1.4
```

Of course you may use different pathnames, depending on where you’ve installed Tomcat or Java.

❷ For expected heavy loads, you might want to increase the memory given to the server. If so, edit `$TOMCAT_HOME/bin/tomcat.sh` and set the desired amount of memory, for example:

```
TOMCAT_OPTS = "-Xmx512m"
```

❸ Edit `$TOMCAT_HOME/conf/server.xml` and add the following:

```
<Context path="/dodsC"
         docBase="webapps/dodsC"
         crossContext="false"
         debug="0"
         reloadable="false" >
</Context>
```

❹ Create the directory `$TOMCAT_HOME/webapps/dodsC`, and copy dodsC.war (find it at ftp://ftp.unidata.ucar.edu/pub/thredds/dodsC.war) into it.

❺ Unpack dodsC.war:

```
cd $TOMCAT_HOME/webapps/dodsC
jar -xf dodsC.war
```

❻ Default server values are in the web.xml file, located in `$TOMCAT_HOME/webapps/dodsC/WEB-INF/`. You might want to change one or more of the `init-param` values, but it should work correctly right out of the box. In particular, runtime debug flags can be set with an init-param name of `DebugOn`, and a value of (space delimited) debug flag names. Currently only the `showRequests` flag is useful to the user. This will log each request received by Tomcat to the server console.

Here's an excerpt from an example `web.xml` file:

```
<web-app>
  <display-name> OPeNDAP Catalog/Aggregation Server</display-name>
  <description>
    OPeNDAP Catalog Aggregation Server - for netCDF and
    Aggregation datasets.
  </description>
  <servlet>
    <servlet-name> dodsC </servlet-name>
    <servlet-class>
      dods.servers.agg.CatalogServlet
    </servlet-class>
    <init-param>
      <param-name>displayName</param-name>
      <param-value>OPeNDAP Aggregation Server</param-value>
    </init-param>
    <init-param>
      <param-name>serverConfig</param-name>
      <param-value>
        file:///../webapps/dodsC/AggServerConfig.xml
      </param-value>
    </init-param>
    <init-param>
      <param-name>maxDODSDatasetsCached</param-name>
      <param-value>0</param-value>
    </init-param>
    <init-param>
      <param-name>maxNetcdfFilesCached</param-name>
      <param-value>100</param-value>
    </init-param>
    <load-on-startup/>
  </servlet>
  <servlet-mapping>
    <servlet-name>dodsC</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>30</session-timeout>
    <!-- 30 minutes -->
  </session-config>
</web-app>
```

2.2 Locating the Configuration File

Now you have an Aggregation Server. But it won't work until you configure it.

If you've successfully completed the steps above, the aggregation server is now installed. However, though you've successfully configured the Tomcat servlet engine to run the Aggregation Server, you haven't yet configured the Aggregation Server itself. To do that, you need to create and install a configuration file. There is an example configuration file, `AggServerConfig.xml`, provided in the Aggregation Server distribution. Look for it in `$TOMCAT_HOME/webapps/dodsC/WEB-INF/` (it comes in the `dodsC.war` archive file). Copy this file up to `$TOMCAT_HOME/webapps/dodsC` then modify it for your datasets. Chapter 3 contains instructions for comfiguring your server.

If you want to keep the configuration file somewhere else (in a different directory or on a different machine, you must modify Tomcat's `web.xml` file. The Aggregation Server configuration file location is specified by the `serverConfig` parameter. For example, the following XML code specifies that the configuration file may be found at `http://www.opendap.net/aggser.xml`

```
<init-param>
  <param-name>serverConfig</param-name>
  <param-value>http://www.opendap.net/aggser.xml</param-value>
</init-param>
```

You can also use an absolute path to locate the configuration file on your local machine. Write it like this for a Win machine:

```
<init-param>
  <param-name>serverConfig</param-name>
  <param-value>file://E:/dev/dodsAS/aggser.xml</param-value>
</init-param>
```

Or like this for a Unix machine:

```
<init-param>
  <param-name>serverConfig</param-name>
  <param-value>file:///usr/local/dodsAS/aggser.xml</param-value>
</init-param>
```

2.3 Setting Cache Sizes

The OPeNDAP Catalog/Aggregation Server uses two independent caches. One is for serving local netCDF files, and the other is used in the process of aggregating remote files.

If your Aggregation Server serves a small number of files, it will work fine with the default settings. If you plan on serving large number of files, you must deal with this issue.

There are two independent caches, one for local netCDF files, and one for remote OPeNDAP files that you are aggregating. Generally, OPeNDAP files take up memory, but no system resources. The default is to set this size to zero, which means there is no limit to the cache. A netCDF file uses a system file handle, so you are limited by maximum number of open files possible on your system. The default is 100.

Set the cache size by adjusting parameters in Tomcat's `web.xml` configuration file (see the web.xml exampleon 18). The file contains a couple of parameter initializations like this:

```
<init-param>
  <param-name>maxDODSDatasetsCached</param-name>
  <param-value>0</param-value>
</init-param>
<init-param>
  <param-name>maxNetcdfFilesCached</param-name>
  <param-value>100</param-value>
</init-param>
```

Here are some tips for selecting an appropriate size for your cache:

- The larger the cache the better for performance. Resources are only used if needed.
- Cache size must be at least as large as the largest number of datasets in a *JoinExisting* aggregation.
- Setting the cache size at least as large as the largest number of datasets in a *JoinNew* or *Union* aggregation helps performance, but is not necessary.

3

Practice II: Configuring the OPeNDAP Catalog/Aggregation Server

Now that you've set up your Aggregation Server, you need to configure it to handle your data. All configuration is done with the XML catalog file described in Chapter 2. This chapter describes what goes into that file.

Before reading this chapter, you should be familiar with basic XML concepts, such as elements, attributes, and Document Type Definition (DTD)'s.

3.1 How it works

The OPeNDAP Catalog/Aggregation Server is configured by adding *metadata* elements of type *aggregation* to catalog *dataset* elements. In the example catalog fragment below (Example 3), a dataset named “NCEP-RUC Model Output” is defined. It will have a relative URL of “NCEP/RUC”. That is, if the Aggregation Server at `http://opendap.org` has a catalog with these entries in it, the aggregated data represented by the *dataset* element will be at `http://opendap.org/NCEP/RUC`. The data a client sees there will be created by joining three NetCDF files along the existing `valtime` dimension, specified in the *variable* attribute of the *aggregation* element. The NetCDF files will be found at `E:/data/070516.nc` and so on.

Specify datasets to be aggregated with XML aggregation elements.

Example 3

```
<dataset name="NCEP RUC Model Output" urlPath="NCEP/RUC">
  <service name="localGrids" serviceType="NetCDF"
    base="file:///E:/data/"/>
  <metadata metadataType="Aggregation">
    <aggregation serviceName="localGrids"
        variable="valtime" type="JoinExisting">
      <fileAccess urlPath="070516.nc"/>
      <fileAccess urlPath="070517.nc"/>
      <fileAccess urlPath="070518.nc"/>
    </aggregation>
  </metadata>
</dataset>
```

The Aggregation Server can also aggregate files from other OPeNDAP servers. In the following example, the aggregation dataset called “Example JoinNew” is created from files that live on the “GSO” OPeNDAP server. The datasets are joined by creating a new dimension called `time`. The data values of that dimension are specified in the *fileAccess* elements. The OPeNDAP datasets out of which this aggregate dataset is constructed have URLs like this: `http://opendap.org/cgi/nph-dsp/htnsst/k2828.htn`.

The aggregated datasets can be remote ones, too.

Example 4

```
<dataset name="Example JoinNew" urlPath="htn_sst_decloud">
  <service name="GSO" serviceType="DODS"
     base="http://opendap.org/cgi/nph-dsp/"/>
  <metadata metadataType="Aggregation">
    <aggregation serviceName="GSO" variable="time"
      type="JoinNew" dateFormat="yyyy/M/d:HH:mm:ss z">
      <fileAccess urlPath="htnsst/k2828.htn" coord="1985/1/1:18:28:28 GMT"/>
      <fileAccess urlPath="htnsst/k1750.htn" coord="1985/1/2:18:17:50 GMT"/>
      <fileAccess urlPath="htnsst/k0718.htn" coord="1985/1/3:18:07:18 GMT"/>
    </aggregation>
  </metadata>
</dataset>
```

The Aggregation Server is also an OPeNDAP server for non-aggregated NetCDF files. The following shows a collection of OPeNDAP datasets. Since they don't have an aggregation *metadata* element (`metadataType=''Aggregation''`, they refer to NetCDF files that are served as OPeNDAP datasets by this Aggregation Server. There is more detail about this in Section 3.4 on page 28.

You can use the Aggregation Server to serve simple netCDF files, too.

Example 5

```
<dataset name="Non-Aggregated Netcdf Files" serviceName="this">
   <property name="internalService" value="local"/>
   <dataset name="RUC/01070516" urlPath="01070516_ruc.nc"/>
   <dataset name="RUC/01070517" urlPath="01070517_ruc.nc"/>
   <dataset name="RUC/01070518" urlPath="01070518_ruc.nc"/>
 </dataset>
```

Note here that the `this` service needs no *service* element to define it.

3.2 Dataset URLs

The *Dataset URL* is the URL that clients use to access an Aggregation Server dataset. This would be `http://opendap.org:8080/RUC/01070516` in Example 5 (assuming that Tomcat is listening to port 8080). For those who are familiar with the operation of other OPeNDAP servers, note that in the Aggregation Server catalog, you do not use the usual OPeNDAP suffixes (`.dds`, `.das`, `.info`, `.html`, and so on). These are automatically added by the Aggregation Server when constructing html pages, and by THREDDS clients when they read the `catalog.xml` file directly.

In general, THREDDS catalogs can specify Datasets that may come from multiple OPeNDAP (and even non-OPeNDAP) servers. For Aggregation Server catalogs, typically we want to specify just Datasets that are served by `this` Aggregation Server. The following fragment shows how a typical Aggregation Server catalog begins:

Example 6

```
<catalog name="Example Agg Server Catalog" version="0.6" >
<dataset name="Top level" dataType="Grid" serviceName="this">
  <service name="this" serviceType="DODS" base=""/>
     ...
    <dataset name="NCEP RUC Model Output" urlPath="NCEP/RUC">
    ...
    </dataset>
</dataset>
</catalog>
```

The first line shows the root *catalog* element. A *catalog* element always has exactly one dataset element, sometimes known as the ”top-level” dataset, shown on the second line. The third line defines the OPeNDAP *service* that is the Aggregation Server itself. By convention, the Aggregation Server in its own catalog is called `this`. The top-level dataset element makes `this` service the default service for all datasets in the catalog. Notice that `this` service has a base consisting of an empty string.

One thing about Aggregation Server catalogs that may be slightly confusing is that a *dataset* element may define an OPeNDAP dataset, or it may just be a container for other dataset elements. In Example 6 on page 25, the dataset element called “Top level” is just a container, because it doesn’t have a *urlPath* attribute or contained *access* elements. The dataset element called ”NCEP RUC Model Output” does define an OPeNDAP dataset, because it has a *urlPath* of `NCEP/RUC`. In this document, we use “Dataset” to mean an OPeNDAP dataset, and “dataset element” to mean an element in the XML document.

The Dataset URL is constructed by concatenating the service *base* and the dataset *urlPath*:

```
Dataset URL = service.base + dataset.urlPath
```

Dataset *urlPaths* should thus be relative to the service, and typically *not* be absolute URLs. By leaving the Aggregation Server `this` service *base* empty, the Dataset URLs become relative to the catalog XML doc itself. The Aggregation Server relies on this, so it’s a very good idea for you to follow this convention. The important things to remember are:

❶ Dataset *urlPaths* should be relative,

❷ dataset *urlPaths* must be unique within an Aggregation Server catalog, and

❸ dataset *urlPaths* for aggregated datasets are arbitrary, since the Aggregation Server is mapping these to the actual files specified in the *aggregation* element. You should use meaningful names however, since the *urlPath* shows up as the dataset name in the DDS.

3.3 Mapping Datasets to Internal Files

The Dataset URL specifies the external name of an OPeNDAP dataset. These are mapped to internal datasets specified in the *fileAccess* elements, which may be other OPeNDAP datasets, or files internally accessible to the Aggregation Server. (For 0.6 these must be NetCDF files, but the Aggregation Server could be extended to handle other types of files.) We will call these internal datasets and files *internal files* , (even though they are not always files) to avoid overloading the word “datasets” any further. The service that the Aggregation Server uses to access these internal files is called the *internal service* , to distinguish from the external service, which is the Aggregation Server itself.

The internal file location is specified by the concatenation of the internal service base and the *fileAccess urlPath*:

```
internal file URL = internalService.base + fileAccess.urlPath
```

The internal service is specified by the *serviceName* attribute of the *aggregation* or *fileAccess* element, as in Example 7. The service elements can be placed in any parent dataset, including the top-level dataset:

Example 7

```
<dataset name="JoinNew Example" urlPath="htnsst">
  <service name="GSO" serviceType="DODS"
      base="http://opendap.org/cgi-bin/nph-dsp/"/>
  <service name="GSO-derived" serviceType="NetCDF"
      base="file:///E:/data/grids/GSO/"/>
  <metadata metadataType="Aggregation">
    <aggregation serviceName="GSO" variable="time" type="JoinNew"
        dateFormat="yyyy/M/d:HH:mm:ss z">
      <fileAccess urlPath="htnsst/1985/k2828.htn"
          coord="1985/1/1:18:28:28 GMT"/>
      <fileAccess urlPath="htnsst/1985/k1750.htn_d.Z"
          coord="1985/1/2:18:17:50 GMT"/>
      <fileAccess serviceName="GSO-derived"
          urlPath="1985/k85003180718.nc"
          coord="1985/1/3:18:07:18 GMT"/>
    </aggregation>
  </metadata>
</dataset>
```

In Example 7, the first two internal files of the aggregation come from the “GSO” OPeNDAP service, and the third from the “GSO-derived” NetCDF service.

3.4 Using the Aggregation Server as a NetCDF server

The Aggregation Server can also serve NetCDF files as OPeNDAP datasets without aggregating them. In this case, specifying the internal service is harder because we don't have an *aggregation* or *fileAccess* element to specify the *serviceName*, and putting a *serviceName* attribute on the dataset element would change the Dataset URL, not the internal file location. To specify a local NetCDF file, use a *property* element whose name is `internalService` and whose value is the name of the internal service to use, as in Example 8.

Example 8

```
...
<service name="local" serviceType="NetCDF"
    base="file:///E:/data/grids/GSO-derived/"/>
<service name="loco" serviceType="NetCDF"
    base="file:///E:/data/grids/UFO-derived/"/>
<dataset name="Non-Aggregated Netcdf Files" serviceName="this">
  <property name="internalService" value="local"/>
  <dataset name="RUC/516" urlPath="516_ruc.nc"/>
  <dataset name="RUC/517" urlPath="517_ruc.nc"/>
  <dataset name="RUC/518" urlPath="518_ruc.nc">
    <property name="internalService" value="loco"/>
  </dataset>
</dataset>
```

In Example 8, the first *property* element sets up a default value for the datasets, while the second *property* tag specifies an exception to the default. The result is that the first two internal files of the dataset come from the "local" NetCDF service, and the third from the "loco" NetCDF service. The only real difference is the base path in which the Aggregation Server looks for the files. Setting up a default in this manner is not essential, but it can save you some typing.

> **NOTE:** When the Aggregation Server is used to serve non-aggregated netCDF files, the *urlPath* of the dataset is used in both the external Dataset URL *and* the internal File URL. Therefore *dataset URLs for NetCDF files are not arbitrary*, since they must map to real files accessible to the Aggregation Server. However the *urlPath* must still be unique within the catalog.

Example 9

```
<catalog name="Example OPeNDAP Aggregation Server Catalog" version="0.6" >
  <dataset name="Top level dataset" dataType="Grid" serviceName="this">
    <service name="local" serviceType="NetCDF" base="file:///MyDataPath"/>
    <property name="internalService" value="local"/>
    ...
  </dataset>
</catalog>
```

3.5 Dataset Aliases

If you want to refer to the same dataset in more than one place in the catalog, use an *alias* attribute, so that you don't have to maintain the aggregation element in more than one place, as in Example 10:

Example 10

```
<dataset name="NCEP RUC Model Output" urlPath="NCEP/RUC" ID="NCEP-RUC">
  <service name="localGrids" serviceType="NetCDF"
    base="file:///E:/data/grids/"/>
  <metadata metadataType="Aggregation">
    <aggregation serviceName="localGrids" varName="valtime"
        aggType="JoinExisting">
      <fileAccess urlPath="01070516_ruc.nc"/>
      <fileAccess urlPath="01070517_ruc.nc"/>
      <fileAccess urlPath="01070518_ruc.nc"/>
    </aggregation>
  </metadata>
</dataset>
...
<dataset name="Another way to sort the datasets>
  <dataset name="NCEP RUC Model Output" alias="NCEP-RUC"/>
  ..
</dataset>
```

3.6 Multiple Aggregation elements

A dataset can contain multiple *aggregation* elements. The Aggregation Server creates a single DDS and DAS by adding the elements of the individual aggregation's DDS and DAS, in the order that they are specified. If a variable or attribute with the same name already exists, then the duplicate is not added. While this does not allow general dataset restructuring, it is useful for simple cases of combining datasets. In the following example:

```
<dataset name="Combine Type 2 and Type 3:" urlPath="Type2and3testCombine">
<metadata metadataType="Aggregation">
<aggregation serviceName="local" varName="time" aggType="JoinExisting">
<fileAccess urlPath="cdc/air.1948.nc"/>
<fileAccess urlPath="cdc/air.1949.nc"/>
<fileAccess urlPath="cdc/air.1950.nc"/>
</aggregation>
<aggregation serviceName="local" aggType="Union">
<fileAccess urlPath="cdc/sst.mnmean.nc"/>
</aggregation>
</metadata>
</dataset>
```

A *JoinExisting* aggregation that has this DDS:

```
Dataset {
  Float32 level[level = 17];
  Float32 lat[lat = 73];
  Float32 lon[lon = 144];
  Float64 time[time = 1096];
  Grid {
    ARRAY:
      Int16 air[time = 1096][level = 17][lat = 73][lon = 144];
    MAPS:
      Float64 time[time = 1096];
      Float32 level[level = 17];
      Float32 lat[lat = 73];
      Float32 lon[lon = 144];
  } air;
} local/MeanAir;
```

and another dataset with this DDS:

```
Dataset {
  Float32 lat[lat = 180];
  Float32 lon[lon = 360];
  Float64 time[time = 233];
  Grid {
    ARRAY:
      Int16 sst[time = 233][lat = 180][lon = 360];
    MAPS:
      Float64 time[time = 233];
      Float32 lat[lat = 180];
      Float32 lon[lon = 360];
  } sst;
} SST;
```

are joined to create the following DDS:

```
Dataset {
  Float32 level[level = 17];
  Float32 lat[lat = 73];
  Float32 lon[lon = 144];
  Float64 time[time = 1096];
  Grid {
    ARRAY:
      Int16 air[time = 1096][level = 17][lat = 73][lon = 144];
    MAPS:
      Float64 time[time = 1096];
      Float32 level[level = 17];
      Float32 lat[lat = 73];
      Float32 lon[lon = 144];
  } air;
  Grid {
    ARRAY:
      Int16 sst[time = 233][lat = 180][lon = 360];
    MAPS:
      Float64 time[time = 233];
      Float32 lat[lat = 180];
      Float32 lon[lon = 360];
  } sst;
} Type2and3testCombine;
```

Note that the sst *Grid* (and maps) are added to the combined dataset, but the top-level variables `lat`, `lon`, and `time` are all taken only from the *JoinNew* dataset, since it was specified first.

4

Configuration Elements and Attributes Specification

The Aggregation Server DTD is a specialization of the THREDDS catalog DTD. This chapter contains a a brief summary of the Aggregation Server DTD followed by a description of the THREDDS catalog DTD.

4.1 Aggregation Server Configuration Elements

The Aggregation Server XML DTD adds four elements to the THREDDS DTD: an *aggregation* element, and the *fileAccess*, *variable*, and *fileScan* elements it depends on.

These two lines appear at the top of the Aggregation Server DTD to include the THREDDS DTD.

```
<!ENTITY % catalogDTD SYSTEM "InvCatalog.0.6.dtd">
%catalogDTD;
```

This is what makes the Aggregation Server a specialization of the THREDDS catalog server.

4.1.1 aggregation

```
<!ENTITY % AggregationType "JoinNew | Union | JoinExisting">
<!ENTITY % VariableType "byte | short | int | float | double | String">

<!ELEMENT aggregation (fileAccess+, variable*, fileScan?)>
<!ATTLIST aggregation
    aggType (%AggregationType;) #REQUIRED
    serviceName CDATA #IMPLIED
    varName CDATA #IMPLIED
    varType (%VariableType;) #IMPLIED
    varUnit CDATA #IMPLIED
    dateFormat CDATA #IMPLIED
>
```

Use the *aggregation* element to define the collections of files to be aggregated, and the manner in which it is to be done.

An *aggregation* element contains one or more *fileAccess* elements, followed by 0 or more *variable* elements, followed by an optional *fileScan* element. (As of Aggregation Server version 0.6, you should not use the *fileScan* element.)

Unlike *Union* and *JoinExist*, *JoinNew* type aggregations must create a new coordinate variable. The variable name is specified with the *varName* attribute to the *aggregation*, and the variable's values must be specified in the *coord* attribute of the *fileAccess* elements contained in this *aggregation* element. The *varType* attribute specifies the type of the new coordinate variable, while the *varUnit* attribute specifies the unit string which is added as its attribute.

For example, consider three data files containing satellite measurements in 1440x720 element arrays. Aggregating them with the following catalog entry:

```
<aggregation serviceName="GSO" aggType="JoinNew" varName="time"
    varType="int" varUnit="secs since 0000-01-01 00:00:00"
    dateFormat="yyyy/M/d:HH:mm:ss z">
  <fileAccess urlPath="qscat/01.dat" coord="0000/1/1:00:00:00 GMT"/>
  <fileAccess urlPath="qscat/02.dat" coord="0000/1/2:00:00:00 GMT"/>
  <fileAccess urlPath="qscat/03.dat" coord="0000/1/3:00:00:00 GMT"/>
</aggregation>
```

will result in this DDS and DAS:

```
Dataset {
  Int32 time[time = 3];
  Byte binarydata[time = 3][latitude = 720][longitude = 1440];
} qscat/bmaps;

Attributes {
  time {
    String units "secs since 0000-01-01 00:00:00";
  }
}
```

and a query on the time variable returns:

```
time[3]
0, 86400, 172800
```

An *aggregation* element appears inside *metadata* elements, which allow content type `ANY`.

These are the possible attributes for an *aggregation* element.

aggType

Required. One of *JoinNew*, *JoinExisting*, or *Union*.

serviceName

The *serviceName* specifies the internal data service to use. The name is given by the *name* attribute of the *service* element. It may be overridden by one of the *fileAccess* elements (or supplied, if it is omitted here). If it is not present in either this element or in the *fileAccess* element, the server will issue an error.

varName

The *varName* specifies the existing (*JoinExisting*) or new (*JoinNew*) coordinate variable to join the files on. It is not used for *Union* type aggregations.

varType

The coordinate values specified in the *coord* attribute of the *fileAccess* elements are converted to the type specified by *varType*, which must be one of *byte*, *short*, *int*, *float*, *double* or *String*. If *varType* is not specified, then the coordinates are added as *Strings*. This attribute is only used in *JoinNew* aggregations.

varUnit

A unit string of the added variable (only for *JoinNew*). This is added to the DAS of the retrieved data, if possible.

dateFormat

Date-valued coordinates are handled in a special way if the *dateFormat* attribute is specified. In this case, the *dateFormat* is the format of the date coordinate values. This format is defined by *java.text.SimpleDateFormat*. The coordinate values are first parsed by *SimpleDateFormat* according to the *dateFormat*. This gives a `long` value in units of "msecs since Jan 1, 1970". If *varUnit* is specified, this value is converted to it using the `ucar.units` package, and the *varUnit* must therefore be convertible with "msecs since Jan 1, 1970". If *varUnit* is not specified, the value is converted into "secs since Jan 1, 1970". If *varType* is specified, the value is converted to that type. If not, the value is converted to a double.

4.1.2 fileAccess

```
<!ELEMENT fileAccess EMPTY>
<!ATTLIST fileAccess
    urlPath CDATA #REQUIRED
    serviceName CDATA #IMPLIED
    coord CDATA #IMPLIED
>
```

The *fileAccess* element specifies a file to be used in this aggregation. The *urlPath* must be specified, and is used with the service to create the internal file's URL. The service is specified through the *serviceName* here or in the parent aggregation element. See Section 3.3 on page 27 for more information.

The *fileAccess* element is analogous to the *access* elements of the THREDDS catalogs, except that it specifies netCDF files or OPeNDAP datasets that are used only internally by the Aggregation Server.

The *coord* is used only by *JoinNew* aggregations, in order to specify the coordinate value that this *fileAccess* corresponds to. The variable type, units, and name are specified with the *aggregation* element.

urlPath

Relative to the base URL given in the *service* element, this is the path to the data file in question.

serviceName

Use this attribute to nominate the service which is to supply this file. If this is omitted, use the *serviceName* nominated by the *aggregation* element. If that one is missing, use `this`.

coord

The *value* of the coordinate variable defined in the *aggregation* element. This is only relevant for *JoinNew* aggregations.

4.1.3 variable

```
<!ELEMENT variable EMPTY>
<!ATTLIST variable
        name CDATA #REQUIRED
>
```

In a *JoinNew* aggregation, all *Grids* will be joined, automatically. Any variables of type *Array* will be joined only if they are specifically nominated by a *variable* element.

name

The name of the *Array* variable to be joined. See Section 1.2.1 on page 12 for an example.

4.1.4 fileScan

NOTE: Do not use this element.

```
<!ELEMENT fileScan EMPTY>
<!ATTLIST fileScan
    urlPath CDATA #REQUIRED
    scanMin CDATA #IMPLIED
>
```

4.2 THREDDS Catalog Configuration Elements

These XML elements are part of the THREDDS catalog server DTD. They are inherited by the OPeNDAP Aggregation Server DTD, which is based on the THREDDS server.

4.2.1 access

```
<!ELEMENT access EMPTY>
<!ATTLIST access
    urlPath CDATA #REQUIRED
    serviceName CDATA #IMPLIED
    serviceType (%ServiceType;) #IMPLIED
>
```

An *access* element specifies how a dataset can be accessed through a data service. It is typically used when there is more than one service available for a dataset.

Typically a *serviceName* is specified, which is the name of a *service* element in a parent element of the same catalog. Note it may not refer to a *service* in another catalog referred to by a *catalogRef* element. The dataset URL is then formed from the service *base* and the access *urlPath*, and optionally the service *suffix* (see Section 3.2 on page 25).

If a *serviceName* is not specified, a *serviceType* must be specified, which creates an ”anonymous service” of that type. In this case the *urlPath* must be absolute.

4.2.2 catalog

```
<!ELEMENT catalog (dataset) >
<!ATTLIST catalog
    name CDATA #REQUIRED
    version CDATA #REQUIRED
    xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
    xmlns CDATA #FIXED "http://www.unidata.ucar.edu/thredds"
>
```

This is the top-level element. A *catalog* element contains exactly one top-level *dataset*. The name of the catalog should be displayed to the user when selecting among catalogs. The *version* allows DTD migration and should be set to `0.6`.

The XLink and default namespaces are declared here, so technically they do not have to be declared in the catalog XML itself. However Internet Explorer cannot deal with namespaces declared in the DTD, so you should add the same two namespace declarations in the catalog element in the XML document itself (see http://www.unidata.ucar.edu/projects/THREDDS/xml/InvCatalog.0.6d.xml

4.2.3 catalogRef

```
<!ELEMENT catalogRef EMPTY>
<!ATTLIST catalogRef
    xlink:type (simple) #FIXED "simple"
    xlink:href CDATA #REQUIRED
    xlink:title CDATA #REQUIRED
>
```

A *catalogRef* element refers to another catalog that becomes a *dataset* inside this catalog. This is used to seperately maintain catalogs and to break up large catalogs. The referenced catalog should not be read until the user explicitly requests it, so that very large dataset collections can be represented with *catalogRef* elements without large delays in presenting them to the user. The referenced catalog is not textually substituted into the containing catalog, but remains a self-contained object. The referenced catalog must be a valid THREDDS catalog, but it does not have to match versions with the containing catalog.

The value of *xlink:href* is the URL of the referenced catalog. The value of *xlink:title* is displayed as the name of the dataset that the user can click on to follow the XLink. Note that the XLink has a fixed type of ”simple” that is part of the DTD, so does not have to be specified in the catalog XML.

The dataset chooser software should seamlessly present a *catalogRef* to the user, for example by eliminating the referenced catalog's top-level dataset in its presentation of the catalog when its name matches the title of the catalogRef title attribute.

4.2.4 dataset

```
<!ENTITY % DataType "Grid | Image | Station">

<!ELEMENT dataset (service*, (documentation | metadata | property)*, access*, (dataset | catalogRe
<!ATTLIST dataset
    name CDATA #REQUIRED
    dataType (%DataType;) #IMPLIED
    authority CDATA #IMPLIED
    ID ID #IMPLIED
    alias IDREF #IMPLIED
    serviceName CDATA #IMPLIED
    urlPath CDATA #IMPLIED
>
```

A *dataset* element represents a logical set of data at a level of granularity appropriate for presentation to a user. A dataset is *selectable* if it contains at least one access path, otherwise it is just a container for nested datasets. If selectable, upon selection, an event is sent to the client software.

A *dataset* element contains 0 or more *service* elements followed by 0 or more *documentation*, *metadata*, or *property* elements in any order, followed by 0 or more *access* elements, followed by 0 or more nested *dataset* or *catalogRef* elements. The data represented by a nested *dataset* element should be a subset, a specialization or in some other sense "contained" within the data represented by its parent *dataset* element.

A dataset must have one or more access paths, specified implicitly through a *urlPath* attribute, or explicitly in contained *access* elements. An access path should be thought of as a URL, but its actually information from which a protocol-aware layer can construct URLs. When there is only one URL, this is typically specified in the *dataset* element itself. When there are multiple URLs, these may be specified in the *dataset* element and/or in contained *access* elements. Multiple URLs specify different services for accessing the dataset. Choices among these different services should be filterered by client software or presented to the user for selection. A URL specified in the dataset element itself is the default URL, which should be the preferred URL when no filtering or user choice is possible. Also see forming URLs.

A dataset may have a *dataType*, specified within itself or in a containing *aggregation*, whose value comes from a controlled vocabulary.

If a *dataset* has an *alias* attribute, the value of the attribute must be an ID of another *dataset* within the same catalog. Note it may not refer to a *dataset* in another catalog referred to by a *catalogRef* element. In this case, any other properties of the dataset are ignored, and the dataset to which the alias refers is used in its place.

A dataset may have a *authority* specified within itself or in a containing *aggregation*. If a dataset has an *ID* and a *authority* attribute, then the combination of the two should be globally unique for all time. If the same dataset is specified in multiple catalogs, then its *authority* - *ID* should be identical if possible.

Many of the properties of a dataset become the default for contained *dataset* elements. This includes *property* elements, and *dataType*, *authority* and *serviceName* attributes. Any *documentation* elements are displayed at the dataset itself when presenting the catalog to the user. Any *metadata* elements apply to all contained datasets.

4.2.5 documentation

```
<!ELEMENT documentation (#PCDATA)>
<!ATTLIST documentation
    xlink:type (simple) #FIXED "simple"
    xlink:href CDATA #IMPLIED
    xlink:title CDATA #IMPLIED
    xlink:show (new | replace | embed) "new"
>
```

A *documentation* element contains or refers to content that should be displayed to an end-user when making selections from the catalog. The content may be HTML or plain text. We call this kind of content ”human readable” information.

The *documentation* element may contain arbitrary plain text content, which should be displayed inline at the position of the *aggregation* or the *dataset* element that contains it.

The *documentation* element may also contain an XLink to an HTML or plain text web page. This text should be either shown inline or displayed when the user activates the XLink, depending on the value of the *xlink:show* attribute, whose default is `new`. If the value of *xlink:show* is `new`, then the content of the XLink should be displayed in a new window when the user selects it. If the value of *xlink:show* is `embed`, then the context should be displayed inline, as if it was text content in the documentation element. If the value of *xlink:show* is `replace`, the content should replace the existing window. The value of *xlink:title* is used for `show` and `replace`, and should be the displayed as the name that the user can click on to follow the XLink. The value of *xlink:show* and *xlink:title* are heuristics for the dataset choosing widget, which may not be able to fully implement them. These heuristics are intended to follow the XLink specification(see http://www.w3.org/TR/xlink/) as closely as possible. Note that the XLink has a fixed type of `simple` that is part of the DTD, so does not have to be specified in the XML.

4.2.6 metadata

```
<!ENTITY % MetadataType "THREDDS | ADN | Aggregation | DublinCore |
        % DIF | FGDC | LAS | Other">

<!ELEMENT metadata ANY>
<!ATTLIST metadata
    xlink:type (simple) #FIXED "simple"
    xlink:href CDATA #IMPLIED
    metadataType (%MetadataType;) #REQUIRED
>
```

A *metadata* element contains or refers to structured information about datasets, which is used by client programs to properly display or search for the dataset. Typically, metadata is not displayed to an end-user when making selections from the catalog, although it may be useful to make it optionally available. We call this kind of content ”machine readable” information.

The *metadata* element must contain a *metadataType* attribute whose value comes from a controlled vocabulary. The types and formats of the metadata are still being developed, and the current list should be considered experimental. Most are currently not operational.

THREDDS a/k/a “Dataset Description”

ADN Alexandria / DLESE format

Aggregation DODS/OPeNDAP Aggregation Server

DublinCore Dublin Core

DIF NASA’s Global Change Master Directory (GCMD) format

FGDC Federal Geographic Data Committee

LAS Live Access Server

The metadata content may be placed in the *metadata* element itself, or it may be pointed to through an XLink, but it may not have both. Generally when the metadata is referenced by an XLink, the information is not read until explicitly requested.

4.2.7 property

```
<!ELEMENT property EMPTY>
<!ATTLIST property
   name CDATA #REQUIRED
   value CDATA #REQUIRED
>
```

Property elements are arbitrary name/value pairs to associate with a dataset, collection or service elements. They will be used to create extended semantics, and should be available to client applications, but not typically displayed during dataset selection. Currently they have no specified semantics.

4.2.8 service

```
<!ENTITY % ServiceType "DODS | ADDE | NetCDF | Catalog | FTP | WMS |
         % WFS | WCS | WSDL | Compound | Other">

<!ELEMENT service (property*, service*)>
<!ATTLIST service
    name CDATA #REQUIRED
    serviceType (%ServiceType;) #REQUIRED
    base CDATA #REQUIRED
    suffix CDATA #IMPLIED
>
```

A *service* element represents a data service. It must contain a *name* and a *serviceType* attribute whose value comes from a controlled vocabulary. It must contain a *name* unique within the catalog (note that catalogs referenced by a *catalogRef* contain their own *ID* namespaces). It must have a *base* attribute and may have an optional *suffix* atribute which are used to construct the dataset URL (see constructing URLS). A *service* element may contain 0 or more *property* elements. These property elements are made available to the application when a dataset is selected, but are not otherwise used.

The scope of a *service* element is its sibling elements and their descendents, excluding catalogs referenced by *catalogRef* elements. The service *name* should be unique within its scope.

A *service* element with *serviceType* equal to `Compound` must have nested service elements, and services with type other than `Compound` may not have nested *service* elements. Nested *service* elements may be used directly by *dataset* or *access* elements. They are at the same scoping level as their parent *service*.

Each *dataset* element must refer to one or more *service* elements that appear in a parent collection. Since typically there will be only a few *service* elements in a catalog but many *dataset* elements, a *service* element factors out the common properties of the data service for efficient representation within the catalog.

A
Acronyms

These acronyms are used frequently in this manual.

DAS Data Attribute Structure A description of the “metadata” attributes for an OPeNDAP dataset. See DODS.

DDS Data Descriptor Structure A description of the data types and sizes of data in an OPeNDAP dataset. See DODS.

DODS Distributed Oceanographic Data System The ancestor to OPeNDAP. See unidata.ucar.edu/packages/dods for information.

DTD Document Type Definition This is the set of definitions that make up an XML specification.

OPeNDAP Open Source Project for Network Data Access Protocol A network protocol for transmitting data across the internet. Though appropriate for many kinds of data, the system was designed with scientific data in mind. See www.nvods.org for information about the group that has developed OPeNDAP and unidata.ucar.edu/packages/dods for information about OPeNDAP’s progenitor, DODS.

THREDDS Thematic Realtime Environmental Data Distributed Services A project of Unidata to create ways to establish useful collections of earth science data. See unidata.ucar.edu/projects/THREDDS for more information.

Index