Building and Testing DMR++ Documents

Table of Contents

1. Intended Audience

This document is for people who want to enable access to HDF and netCDF data files stored in Amazon Web Services (AWS) Simple Storage Service (S3) using the OPeNDAP *Hyrax* data server. It describes how to build the DMR++ documents the Hyrax server uses. This document will also be useful to people who want to build DMR++ documents for other reasons, such as enabling client-side technology like *VirtualiZarr* to access/subset remote data files using only HTTP.

There are some features of the DMR++ builder software that are targeted specifically to the NASA Earth Science Data and Information Systems (ESDIS) project's use of S3 to store data. Look for the *TIP* icon to find those.

2. Introduction

The DMR++ is a metadata file that provides a fast and flexible way to access data stored in Amazon Web Services (AWS) Simple Storage Service (S3), or using any other service that supports HTTP.[1]

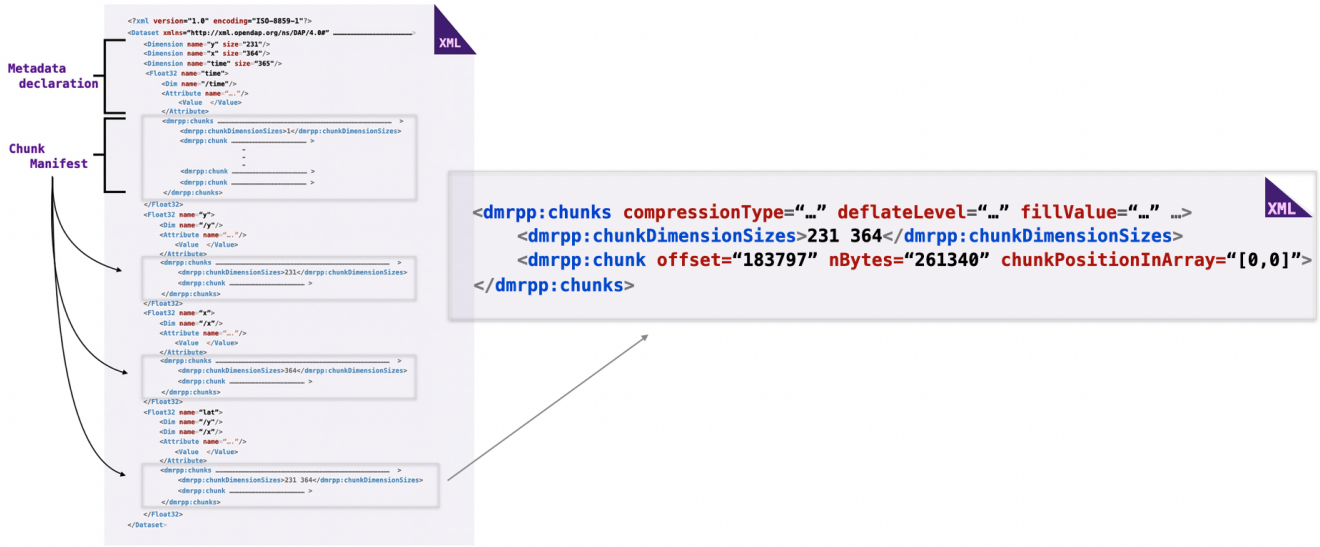

Figure 1. Diagram showing what a DMR++ might look like for a given (individual) file. The DMR++ is an XML document containing metadata declarations of the variables and attributes within the file, and annotated with Chunk Manifests for each Array. Currently, it supports fletcher32, shuffle, and deflate as CompressionType, and deflateLevel between 1 and 9 (9 included).

OPeNDAP developed the DMR++ system in response to the growing need for faster, more efficient access to scientific data stored in large, complex files. As datasets expanded in size and migrated to cloud-based storage systems like S3, traditional methods of reading metadata and retrieving data became increasingly cumbersome and expensive. DMR++ was designed to solve this problem by creating a lightweight, annotated metadata file that describes where data elements are located and how they can be accessed, without requiring full downloads or local processing. By decoupling metadata access from the underlying storage and minimizing data movement, DMR++ enables scalable, cloud-friendly workflows that better support the demands of modern Earth science research.

The DMR++ encodes the location of data residing in a binary data file/object (e.g., an HDF5 file) so that it can be directly accessed, without the need for an intermediate Application Programmer Interface (API) library. The binary data objects may be in a local filesystem, or they may be accessible using HTTP in, for example, an S3 bucket. The Hyrax data server from OPeNDAP can use DMR++ files to provide access and subsetting services for data stored in S3 without first copying their data files from S3. That is, Hyrax can use the DMR++ files to access and subset data 'in place.' This mirrors the behavior of the Hyrax server when used with data files stored on POSIX file systems.

The DMR++ is an extension of the Dataset Metadata Response (DMR) from OPeNDAP's DAP4 protocol. For a full description of DAP4 and the DMR object, see the DAP4 protocol, Sections 1.5.7–1.5.15. The DMR encodes metadata information about the names, data types, and hierarchical relations of the variables that make up a dataset. The DMR++ adds information about the location, size, and other relevant characteristics of those variables. Software can then use this information to read binary data values directly from the dataset's file(s) without using an API library or copying the dataset to temporary storage before accessing the data.

Additional advantages to the DMR++ are:

1. Enables data providers to take advantage of modern storage technologies for large data without having to reformat huge data collections.
2. A DMR++ can be programmatically generated by OPeNDAP software for datasets that are made up of HDF5, NetCDF4, HDF4, and HDF4_EOS2 data files.
3. Data file integrity is preserved.

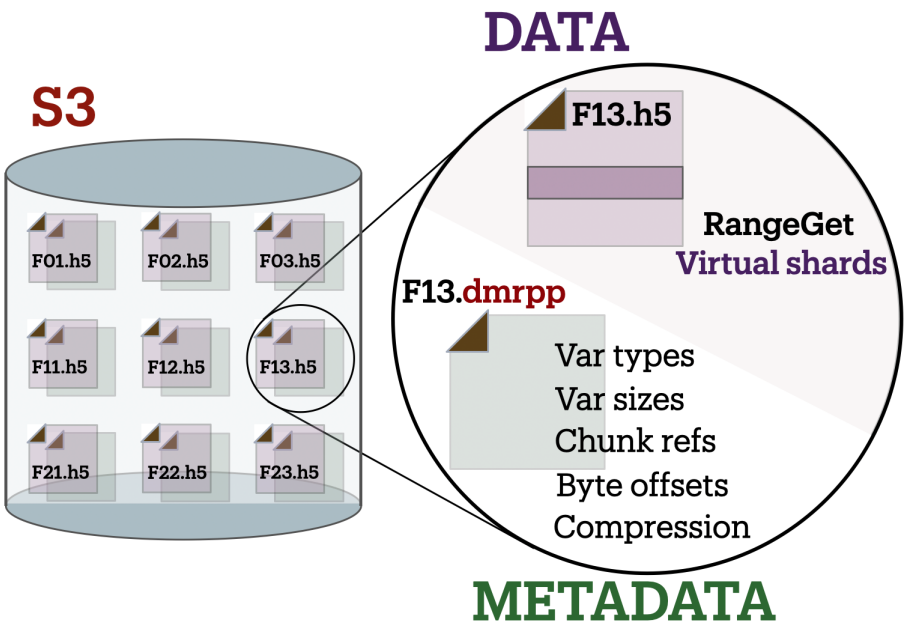

Figure 2. A collection of HDF5 files in an S3 bucket. Each data file has an associated DMR++ file, named using the data file name with the suffix '.dmrpp'. Because the DMR++ uses a URL to reference the source data file, it can be stored 'close' to the data or on a different storage system.

3. Supported Data Formats

The software to build DMR++ documents currently works with HDF5, netCDF4, HDF4, and HDF4-EOS2 files.[2] Other formats like Zarr and netCDF3 are not currently supported by the DMR++ software, but support could be added if requested.

3.1. The Gory Details

Technologies such as HDF5 are best characterized as tools for defining *self-describing* data files. These files are widely adopted in scientific domains because they support a diverse range of organizational structures for information. In the case of NASA ESDIS, nearly all the more than 8,600 data collections (encompassing over one billion individual files) define distinct sets of *variables*, effectively making each collection a unique data format. Despite these differences, a small number of API libraries can be used to consistently access the data across all collections.

While we aim to provide support for all possible HDF5, HDF4, etc., data files, there are aspects of the *data models* these API libraries implement that the current DMR++ software does not cover. As of April 2025, support for HDF5, as it is used by the NASA ESDIS collections, is close to complete. The best approach to determining if the OPeNDAP DMR++ builder software will work for a given collection is to try it. We suggest picking one or two granules/files and then following the steps outlined here in Section 4 followed by the testing process described in Section 5. Are the variables all present? Are the values, or a sampled subset of values, correct?

Support for HDF4 and HDF4-EOS2 data files is much newer, and more work will need to be done on edge cases than for HDF5. However, as of April 2025, the same advice applies to these as to the HDF5 case. Try to build the DMR++ and then test the result.

In NASA collections using HDF4-EOS2, geolocation information is often not included within individual data files. This approach minimizes storage requirements by avoiding the repeated storage of redundant information. For instance, a MODIS collection may contain approximately 10,000 files (granules), each referencing geolocation data drawn from a common set of around 120 predefined global regions. To manage this, DMR++ generates and stores the geo-referencing information in additional compressed data files, but without an attempt to limit that to the minimum amount of the geo-referencing data. Efforts to optimize the storage of HDF4-EOS2 geo-referencing data are planned and will be prioritized based on user demand.

3.1.1. Is My NetCDF File A Version *3* or Version *4* File?

OPeNDAP's DMR++ software does not currently support netCDF3 files.[3] A complicating factor in building DMR++ documents is that it can be hard to tell at a glance if a file is netCDF version 3 or version 4. A file with the suffix *.nc4* is conventionally recognized as a *netCDF-4* file. However, the file suffix *.nc* is ambiguous since it is often used for both *netCDF-3* and *netCDF-4* files.

You can use the `ncdump` command to determine if a *netCDF* file is either classic *netCDF-3* or *netCDF-4* (You can learn more in the NetCDF documentation here). Here are two files, both using the suffix

`.nc` where the first is netCDF3 and the second is netCDF4.

```
% ncdump -k fnoc1.nc
classic

% ncdump -k SMAP_L4_SM_aup_20150420T210000_Vv7032_001.nc
netCDF-4
```

4. How to Build DMR++ reference files

4.1. Assumptions

You have:

- Docker installed on your computer and at least a basic understanding of its use.
- Data files in a directory on your computer

In the following, `%` is the terminal prompt. Only some commands produce output, and for those that do, the output is shown below the command. The paths, etc., on your computer will almost certainly be different.

4.2. Examples

In this section we jump right into some examples without much explanation. This shows the minimum amount of work needed to build the DMR++ and sidecar files. See Section 4.5 for details about the `gen_dmrpp_side_car` command, which is the recommended command for building DMR++ documents (April 2025).

Change to the directory that holds your data files and assign an environment variable to the full pathname of that directory. This will streamline some of the later steps in this section. In my case that directory is called `HDF4-dir`, and I used the environment variable 'DATA.'

```
% cd HDF4-dir
% export DATA=$(pwd)
% echo $DATA
/Users/jimg/src/opendap/hyrax_git/HDF4-dir
```

Here are the files on my computer in the directory assigned to $DATA

```
% ls
3B42.19980101.00.7.HDF
3B42.19980101.03.7.HDF
3B42.19980101.06.7.HDF
3B42.19980101.09.7.HDF
```

```
3B42.20130111.06.7.HDF
3B42.20130111.09.7.HDF
AIRS.2009.01.01.L3.RetStd_IR001.v7.0.3.0.G20160024306.hdf
AIRS.2009.01.02.L3.RetStd_IR001.v7.0.3.0.G20160024358.hdf
AIRS.2009.01.03.L3.RetStd_IR001.v7.0.3.0.G20160024538.hdf
AMSR_E_L2_Land_V09_200206191023_D.hdf
AMSR_E_L2_Land_V09_200206191112_A.hdf
AMSR_E_L3_SeaIce25km_V15_20020601.hdf
MCD12Q1.A2022001.h10v06.061.2023243073808.hdf
MCD19A1.A2024025.h10v06.061.2024027100206.hdf
MOD10A1F.A2024025.h01v08.061.2024027134335.hdf
MOD10A1F.A2024025.h01v09.061.2024027130238.hdf
MOD10A1F.A2024025.h01v10.061.2024027131939.hdf
MOD11A1.A2024025.h10v06.061.2024028004317.hdf
```

Run the Docker container. The docker run command returns the Container ID (a long hexadecimal string) when the `-d` (run a detached container) is used. The `--name` option sets *hyrax* as the name of the container which will be used in later commands. Running the container this way enables us to use both build DMR++ documents and later test them.

```
% docker run -d -h hyrax -p 8080:8080 -v $DATA:/usr/share/hyrax --name=hyrax
opendap/hyrax:1.17.1-126
9c88a0d4abe55f17802afd81150280073314f3940b9cd4973ea60dbc43f733a9
```

In this document, we use an explicit version number when we show the container being used. We do that to make sure that the information here is repeatable. In practice, you can replace that version number with the word *snapshot* to get the most recent version of the command (and the most recent bundled Hyrax server). That is, where we use `opendap/hyrax:1.17.1-126` using `opendap/hyrax:snapshot` instead will get the most recent version of the software.

Do not confuse the Docker tag *snapshot* with *latest*. In all but the most unusual situations, you do **NOT** want the container tagged *latest*. Use the tag *snapshot*.

If you want to use the latest version of the `gen_dmrpp_side_car` command, replace the version number in *opendap/hyrax:1.17.1-126* with *snapshot*. Using *opendap/hyrax:snapshot* will always get the most recent version of the software.

To build a DMR++ for the first AIRS file we can run the `gen_dmrpp_side_car` command, using `docker exec`, with the file's name. Because this file is an HDF4 file, the command option `-H` is used.

Building a DMR++ for an AIRS HDF4 file/granule.

```
% docker exec -it -w /usr/share/hyrax hyrax gen_dmrpp_side_car -i
AIRS.2009.01.01.L3.RetStd_IR001.v7.0.3.0.G20160024306.hdf -H -U

% ls
...
```

```
3B42.20130111.09.7.HDF
AIRS.2009.01.01.L3.RetStd_IR001.v7.0.3.0.G20160024306.hdf
AIRS.2009.01.01.L3.RetStd_IR001.v7.0.3.0.G20160024306.hdf.dmrpp
AIRS.2009.01.02.L3.RetStd_IR001.v7.0.3.0.G20160024358.hdf
...
```

In this second example both the DMR++ and a sidecar *missing data* file (`3B42.19980101.00.7.HDF_mvs.h5`) are built. As is often the case, the DMR++ and missing data files together are only 2% of the data file's size.

Even though the input data file was an HDF4-EOS2 file, the missing data file uses HDF5 to store the values.

This is also an HDF4 file, so the `-H` option is used.

Building both the DMR++ and a missing data file

```
% docker exec -it -w /usr/share/hyrax hyrax gen_dmrpp_side_car -i
3B42.19980101.00.7.HDF -H -U

% ls -l
total 1245840
-rw-r--r--@ 1 jimg  staff     774595 Aug 22  2024 3B42.19980101.00.7.HDF
-rw-r--r--  1 jimg  staff       6514 Apr 21 22:42 3B42.19980101.00.7.HDF.dmrpp
-rw-r--r--  1 jimg  staff       8075 Apr 21 22:42 3B42.19980101.00.7.HDF_mvs.h5
-rw-r--r--@ 1 jimg  staff     765742 Aug 22  2024 3B42.19980101.03.7.HDF
 ...
```

The final example in this section shows building a DMR++ for an HDF5 file. For an HDF5 file, do not include the `-H` option.

Build a DMR++ for an HDF5 file.

```
% docker exec -it -w /usr/share/hyrax hyrax gen_dmrpp_side_car -i
SMAP_L4_SM_aup_20150420T210000_Vv7032_001.h5 -U

% ls -l
total 1895576
 ...
-rw-r--r--@ 1 jimg  staff   95114159 Aug  5  2024
SMAP_L4_SM_aup_20150420T210000_Vv7032_001.h5
-rw-r--r--  1 jimg  staff     277290 Apr 25 15:51
SMAP_L4_SM_aup_20150420T210000_Vv7032_001.h5.dmrpp
```

4.3. How DMR++ References the Matching Data File

This section is primarily for NASA ESDIS users of the DMR++ document builder. However, there is some generally useful information here, so most readers should

> skim it over.

A DMR++ document is an eXtensible Markup Language (XML) document. We call the data file/granule that the DMR++ describes the *source data file*. Each DMR++ has at least one source data file, but may have more than one source data file, for example, with HDF4-EOS2 geo-referencing data. The first XML *element* in the DMR++ contains a URL that points to the DMR++ document's source data file. It looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Dataset xmlns="http://xml.opendap.org/ns/DAP/4.0#"
    xmlns:dmrpp="http://xml.opendap.org/dap/dmrpp/1.0.0#" dapVersion="4.0" dmrVersion
="1.0"
    name="SMAP_L4_SM_aup_20150420T210000_Vv7032_001.h5"
    dmrpp:href=
"https://test.opendap.org/examples/SMAP_L4_SM_aup_20150420T210000_Vv7032_001.h5"
    dmrpp:version="3.21.1-243">
```

There are three *XML attributes* in the root element of the DMR++ that are relevant to this discussion. They are:

```
name="SMAP_L4_SM_aup_20150420T210000_Vv7032_001.h5"
dmrpp:href="https://test.opendap.org/examples/SMAP_L4_SM_aup_20150420T210000_Vv7032_00
1.h5"
dmrpp:version="3.21.1-243">
```

name	The name of the data file/granule.
dmrpp:href	The full URL to the source data file.
dmrpp:version	The version of the DMR++ builder software used to make this DMR++ document.

The value of the `dmrpp:href` attribute is the source of data values that the Hyrax data server will use with building data responses. This URL can be either an HTTP, HTTPS or *file://* URL (for more about the latter option, see Section 5.

However, when the OPeNDAP DMR++ was first developed for use by NASA ESDIS, we did not want to encode the URL to the data file into the DMR++. Instead, we planned on using the ESDIS Common Metadata Repository (CMR) to look up information about a granule and use that to find the source data file. This helped guard against having to edit many of the documents while the ESDIS system was in flux (i.e., it was a decision well aligned with agile development principles). In place of an explicit URL to the source data file, the `gen_dmrpp_side_car` will, by default, use a template string that the hyrax data server substitutes at runtime with the current data source URL as read from CMR.

What if you do not need or want that? The `-u` option of `gen_dmrpp_side_car` provides a way to tell the DMR++ document builder to use a specific value for the data source URL. The following examples show the DMR++ XML *with* the template value for the data source URL and then using a URL set

with the -u option.

With the template

```
% docker exec -it -w /usr/share/hyrax hyrax gen_dmrpp_side_car -i
SMAP_L4_SM_aup_20150420T210000_Vv7032_001.h5
%head  SMAP_L4_SM_aup_20150420T210000_Vv7032_001.h5.dmrpp
<?xml version="1.0" encoding="ISO-8859-1"?>
<Dataset xmlns="http://xml.opendap.org/ns/DAP/4.0#"
xmlns:dmrpp="http://xml.opendap.org/dap/dmrpp/1.0.0#" dapVersion="4.0"
dmrVersion="1.0"
    name="SMAP_L4_SM_aup_20150420T210000_Vv7032_001.h5"
    dmrpp:href="OPeNDAP_DMRpp_DATA_ACCESS_URL"
    dmrpp:version="3.21.1-243">
```

The template value for the data source URL is OPeNDAP_DMRpp_DATA_ACCESS_URL

Explicit data source URL, set using -u

```
% docker exec -it -w /usr/share/hyrax hyrax gen_dmrpp_side_car -i
SMAP_L4_SM_aup_20150420T210000_Vv7032_001.h5 -u
https://test.opendap.org/examples/SMAP_L4_SM_aup_20150420T210000_Vv7032_001.h5
% head SMAP_L4_SM_aup_20150420T210000_Vv7032_001.h5.dmrpp
<?xml version="1.0" encoding="ISO-8859-1"?>
<Dataset xmlns="http://xml.opendap.org/ns/DAP/4.0#"
xmlns:dmrpp="http://xml.opendap.org/dap/dmrpp/1.0.0#" dapVersion="4.0"
dmrVersion="1.0"
    name="SMAP_L4_SM_aup_20150420T210000_Vv7032_001.h5"

dmrpp:href="https://test.opendap.org/examples/SMAP_L4_SM_aup_20150420T210000_Vv7032_00
1.h5"
    dmrpp:version="3.21.1-243">
```

The -u option provides the literal text for the value of the dmrpp:href XML attribute.

4.4. How DMR++ References a Sidecar File for Geo-referencing Data

The mechanism described above for the data source URL, where the DMR++ builder provides a template value unless overridden using the -u option, applies similarly to references for sidecar geo-referencing data. By default, the name of the sidecar file itself is used. To include a template value (OPeNDAP_DMRpp_SC_DATA_ACCESS_URL) instead, the -U option must be specified. As with the data source URL, the -s option (described below in Section 4.5) may be used to explicitly set the sidecar file URL.

There is one exception to the rule that -u is used for the data source URL and -s is used for the sidecar data file. If -u is used, that name will be used as a *pattern* for the sidecar data file such that the missing data file will be assumed to be named the same as the data source, but with the suffix

_mvs.h5.

In this example, we show the three files made from an HDF4-EOS2 file that where the sidecar file is necessary. The output of the command is shown first, followed by two views inside the DMR++ document.

An Explicit Data Source URL is a Pattern for an Explicit Sidecar Data URL

```
% docker exec -it -w /usr/share/hyrax hyrax gen_dmrpp_side_car -i
3B42.20190110.06.7.HDF -H -u file:///usr/share/hyrax/3B42.20190110.06.7.HDF

% ls -l
total 1895672
-rw-r--r--@ 1 jimg  staff     600255 Aug 22  2024 3B42.20190110.06.7.HDF
-rw-r--r--  1 jimg  staff       6595 Apr 25 17:21 3B42.20190110.06.7.HDF.dmrpp
-rw-r--r--  1 jimg  staff       8075 Apr 25 17:21 3B42.20190110.06.7.HDF_mvs.h5
```

The Resulting XML, edited. Look for the file:/// URLs marked with the comments HERE.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Dataset xmlns="http://xml.opendap.org/ns/DAP/4.0#" xmlns:dmrpp=
"http://xml.opendap.org/dap/dmrpp/1.0.0#"
    dapVersion="4.0" dmrVersion="1.0"
    name="3B42.20190110.06.7.HDF"
    dmrpp:href="file:///usr/share/hyrax/3B42.20190110.06.7.HDF">
<!-- HERE -->
    <Dimension name="nlon" size="1440"/>
    <Dimension name="nlat" size="400"/>
    <Float32 name="nlat">
        ...
        <dmrpp:chunks compressionType="deflate" deflateLevel="4" fillValue="0"
byteOrder="LE">
            <dmrpp:chunkDimensionSizes>400</dmrpp:chunkDimensionSizes>
            <dmrpp:chunk offset="5435" nBytes="636" chunkPositionInArray="[0]"
                href="file:///usr/share/hyrax/3B42.20190110.06.7.HDF_mvs.h5" />
<!-- HERE -->
        </dmrpp:chunks>
        ...
```

4.5. Explanation of the gen_dmrpp_side_car Command Options

The gen_dmrpp_side_car command takes a few options that control how it builds DMR++ and sidecar files.

-i The -i option is used to name the *input data file*. This data file should be found in the directory where the command is being run, or one of its child directories. In the latter case, the relative pathname to the file should be used. This option is

required.

-H The `-H` option tells the command that the input file is an HDF4 or HDF4-EOS2 data file. If the `-H` option is not used, then the data file is assumed to be either HDF5 or netCDF4.

-c The `-c` option results in DMR++ and sidecar files that follow the Climate Forecast (CF) conventions. Using this option provides a DMR++ that mimics the behavior of the Hyrax server when it is used to serve data stored on POSIX file systems with the *EnableCF* option turned on. This organizes the presentation of the variables to follow CF and flattens the internal hierarchy of the data files, hiding any *Groups*.

-D The `-D` option will disable the build of a sidecar file, even when one would normally be required. The default is to build sidecar data files when needed.

-U Use the template value (`OPeNDAP_DMRpp_SC_DATA_ACCESS_URL`) for the value of the sidecar data file URL. The default is to use only the name of the template file. In most cases, if a sidecar file is made the `-U` or `-u <URL>` options should be used.

-u/--URL The `-u/--URL` and `-s/--SURL` options control how URLs are represented in the DMR++ document. It is possible to build a DMR++ before the location of the data file in S3, for example, is known. In this case, the URL that references the data file will be represented by a 'template' value and substituted into the DMR++ *when the document is used*, nominally by the Hyrax service at runtime (although other software can also do this substitution - it is a simple text replacement). See Section 4.3. If this option is used, no run-time substitution of the data source URL will be performed.

-s/--SURL The `-s/--SURL` option provides the same feature for the URL that references the sidecar geo-referencing data file. The Hyrax service *assumes* that the data file URL can be determined by removing the suffix `.dmrpp` from the DMR++ URL. Similarly, it assumes that the sidecar data file URL can be found by replacing the `.dmrpp` suffix with `_mvs.h5`. See Building both the DMR++ and a missing data file. Note that these options can be used to provide real values for data file and sidecar data URls. In that case, the given values will be used in the DMR++ instead of the template values. No run-time substitution of the URLs will be performed.

4.6. Explanation of the `docker run` Command Options

In the Section 4.2 we used one docker command to start a container and then a second docker command to run the DMR++ builder inside that container. Here is an explanation of those commands in more detail. First, the container is started on the host computer.

```
% docker run -d -h hyrax -p 8080:8080 -v $DATA:/usr/share/hyrax --name=hyrax
opendap/hyrax:1.17.1-126
9c88a0d4abe55f17802afd81150280073314f3940b9cd4973ea60dbc43f733a9
```

The `docker run -d ⋯` command will run the Hyrax container on your computer (called the *host*

computer) in *detached* mode. The Hyrax container includes both the complete Hyrax service and the `gen_dmrpp_side_car` command. Later this server will be used to test the DMR++ documents that are built.

The volume mount, from `$DATA` to `/usr/share/hyrax` mounts the current directory of the host computer running the container to the directory */usr/share/hyrax* inside the container. That directory is the root of the Hyrax server's data tree. This means that the data files in the `$DATA` directory will be accessible by the server running in the container without any other configuration.

Complete option summary:

-d, --detach	Run container in the background and print container ID
-h, --hostname	Set the container's host name
-p, --publish	Publish a container's port(s) to the Docker host
-v, --volume	Mount a volume so that the container can use files on the Docker host
--name	Assign a name to the container; this name can be used in later Docker commands

Once running, the container is used to run the command that will build the DMR++ document.

```
% docker exec -it -w /usr/share/hyrax hyrax gen_dmrpp_side_car -i
3B42.19980101.00.7.HDF -H -U
```

The command that built the DMR++ (and sidecar) file really consists of *two commands*. The first is `docker exec -it -w /usr/share/hyrax hyrax` which instructs docker to *execute* a program in the running container named *hyrax* and do so by first changing to the directory */usr/share/hyrax* in that container. By using the `-w` option we are able to run the gen_dmrpp_side_car command in the directory within the container where data appear.

The second command instructs the docker container to run `gen_dmrpp_side_car` using the arguments `-i 3B42.19980101.00.7.HDF -H -U` which mean use the file *3B42.19980101.00.7.HDF* as the input data file, assume it is an HDF4 file and use the template name for the sidecar data file.

Complete option summary for the `docker exec` command:

-i, --interactive	Set the working directory inside the container
-t, --tty	Allocate a pseudo-terminal
-w, --workdir	Set the working directory inside the container

5. Testing DMR++ Containers Using the Builtin Hyrax Server

One of the more confounding things about testing DMR++ documents is that it requires a data server, or some software component, that can interpret the documents. Instead of the data being directly available, the DMR++ sits between the software and the data. In this section we show how to test a DMR++ document that using the Hyrax server running in the container used to build the DMR++ document. To do this, we will build the DMR++ with *file URLs* for the data and sidecar files instead of *HTTP URLs* or the *template values* that the command would normally use.

```
% docker exec -it -w /usr/share/hyrax hyrax gen_dmrpp_side_car -i
3B42.20130111.09.7.HDF -H -u 'file:///usr/share/hyrax/3B42.20130111.09.7.HDF'
```

Copy that pattern for whatever file you use. From the `/usr/share/hyrax` directory, you pass *gen_dmrpp_side_car* the name of the file (because it's local to the current directory) using the `-i` option. The `-u` option tells the command to embed the URL that follows it in the DMR++. I've used a *file://* URL to the file */usr/share/hyrax/3B42.19980101.00.7.HDF*.

In the URL above, three slashes follow the colon: two from the way a URL names a protocol and one because the pathname starts at the root directory.

Let's look at how the *hyrax* service will treat that data file using the DMR++. In a browser, go to http://localhost:8080/opendap/. The *hyrax* container must be started using the `docker run` command for this to work (Section 4.2).

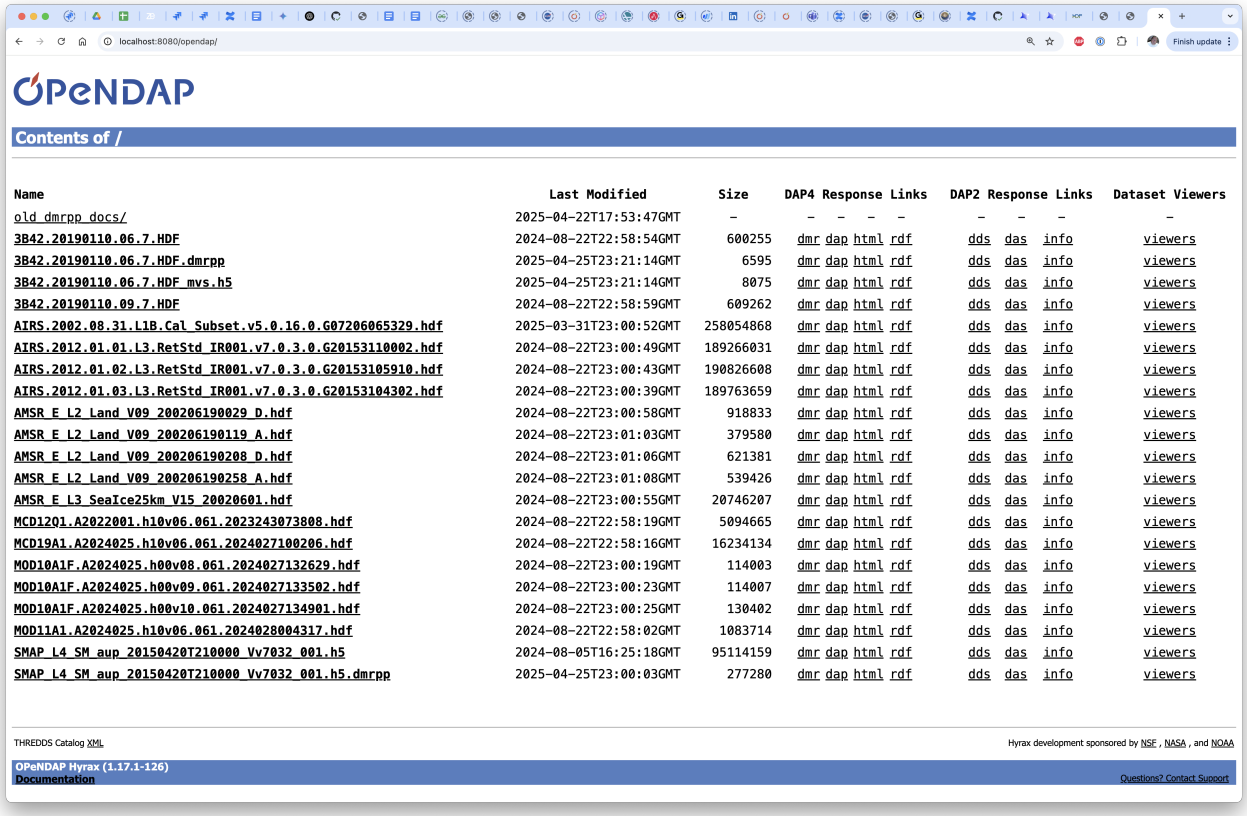

Figure 3. Hyrax Catalog view of all files available.

The server caches data catalog information for 5 minutes (although this can be configured) so new items (e.g., DMR++ documents) may not show up right away. To force the display of a DMR++ that you just created, click on the source data file name and edit the URL so that the suffix `.dmr.html` is replaced by `.dmrpp.dmr` .

Click on your equivalent of the `3B42.20130111.09.7.HDF` link, subset, download, and open in Panoply or the equivalent.

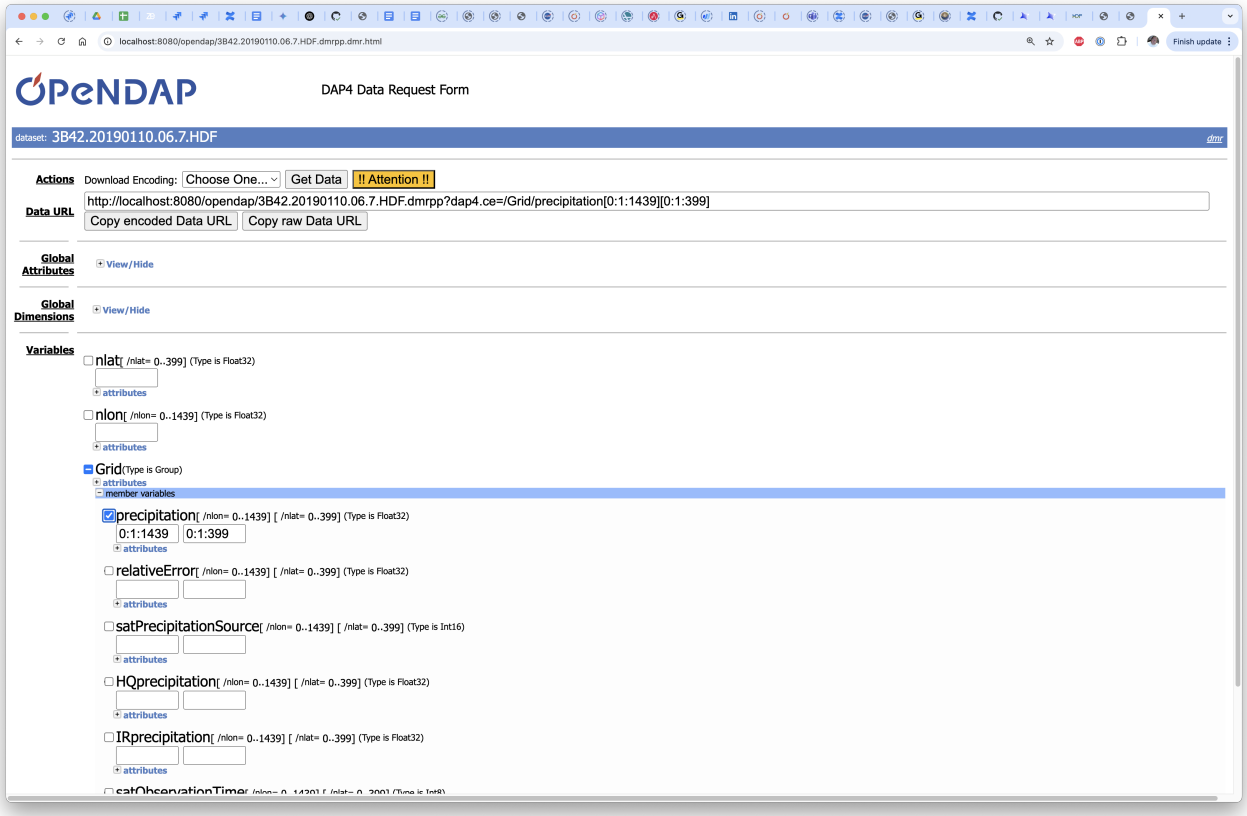

Figure 4. Page view of the DAP Data Request Form for subsetting the dataset.

You can run batch tests in lots of files by building many DMR++ documents and then asking the server for various responses (*nc4*, *dap*) from the DMR++ and the original file. Those could be compared using various schemes such as the command *getdap4* included in the container. The `getdap4` command can be used to compare the *dap* responses from the data file and the DMR++ document.

Below is a comparison of the same underlying data, the left window shows the data returned using the DMR++, the right shows the data read directly from the file using the server's builtin HDF4 reader.

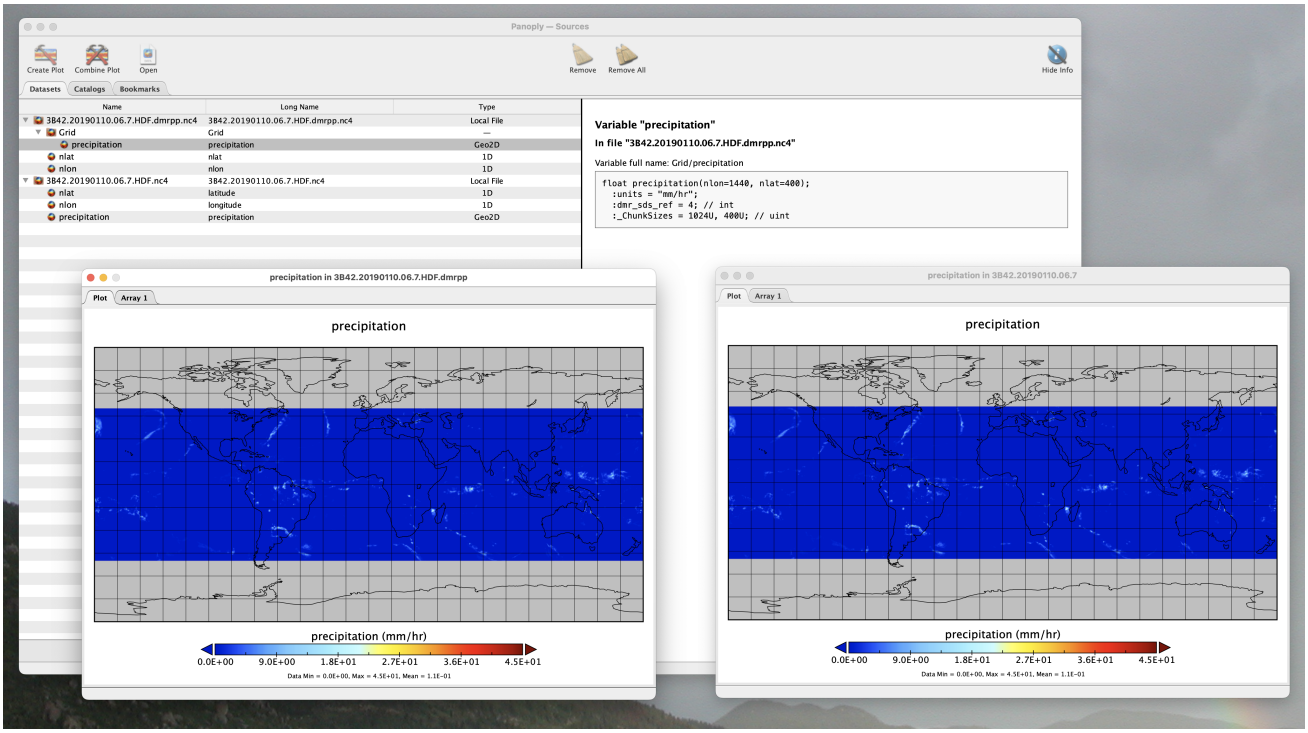

Figure 5. Comparison of responses from a DMR++ (left) and the native file handler (right).

6. Serving Data Using DMR++ Files

This is older text that repeats some of the above material, but it provides a good reference for using the DMR++ in a range of data provider situations.

There are three fundamental deployment scenarios for using DMR++ files to serve data with the Hyrax data server.

This can be simply categorized as follows: The DMR++ file(s) are XML files that contain a root `dap4:Dataset` element with a `dmrpp:href` attribute whose value is one of:

1. A http(s):// URL referencing to the underlying granule files via http.
2. A file:// URL that references the granule file on the local filesystem in a location that is inside the BES' data root tree.
3. The template string `OPeNDAP_DMRpp_DATA_ACCESS_URL`

Each will be discussed in turn below.

By default, Hyrax will automatically associate files whose name ends with ".dmrpp" with the **DMR++** handler.

6.1. Using DMR++ with HTTP/S URLs

If the DMR++ files that you wish to serve contain `dmrpp:href` attributes whose values are http(s) URLs then there are 2+1 steps to serve the data:

1. Place the DMR++ files on the local disk inside the directory tree identified by the

`BES.Catalog.catalog.RootDirectory` in the BES configuration.

2. Ensure that the Hyrax `AllowedHosts` list is configured to allow Hyrax to access those target URLs. This can be done by adding new regex records to the `AllowedHosts` list in `/etc/bes/site.conf`, creating that file as need be.
3. If the data URLs require authentication to access, then you'll need to configure Hyrax for that too. See The Hyrax Data Server Installation and Configuration Guide for more information.

6.2. Using DMR++ with file URLs

Using DMR++ files with locally held files can be useful for verifying that DMR++ functionality is working without relying on network access that may have data rate limits, authenticated access configuration, or security access constraints. Additionally, in many cases the DMR++ access to the locally held data may be faster than through the native `netcdf-4/HDF5` data handlers.

To use DMR++ files that contain `file://` URLs:

1. Place the DMR++ files on the local disk inside the directory tree identified by the `BES.Catalog.catalog.RootDirectory` in the BES configuration.
2. Ensure that the DMR++ files contain only file:// URLs that refer to data granule files that are inside the directory tree identified by the `BES.Catalog.catalog.RootDirectory` in the BES configuration.

Note: For Hyrax, a correctly formatted file URL must start with the protocol `file://` followed by the full qualified path to the data granule, for example:

`/usr/share/hyrax/ghrsst/some_granule.h5`

so that the completed URL will have three slashes after the first colon:

`file:///usr/share/hyrax/ghrsst/some_granule.h5`

6.3. Using DMR++ with the template string (NASA).

This is most relevant to the operation of the NASA ESDIS Hyrax in the Cloud server deployment.

Another way to serve DMR++ files with Hyrax is to build the DMR++ files **without** valid URLs but with a template string that is replaced at runtime. If no target URL is supplied to *get_drmpp* at the time that the DMR++ is generated the template string: `OPeNDAP_DMRpp_DATA_ACCESS_URL` will be added to the file in place of the URL. The at runtime it can be replaced with the correct value.

Currently, the only implementation of this is Hyrax's NGAP service that, when deployed in the NASA NGAP cloud, will accept *REST_* URLs that are defined as having a URL path component with two mandatory and one optional parameters:

```
MANDATORY: "/collections/UMM-C:{concept-id}"
MANDATORY: "/granules/UMM-G:{GranuleUR}"
```

Example Hyrax in the Cloud REST URL

```
https://opendap.earthdata.nasa.gov/collections/C1443727145-
LAADS/granules/MOD08_D3.A2020308.061.2020309092644.hdf.nc
```

UMM-C:{concept-id} /collections/C1443727145-LAADS

UMM-G:{GranuleUR} /granules/MOD08_D3.A2020308.061.2020309092644.hdf.nc

When encountering this type of URL, Hyrax will decompose it and use the content to formulate a query to the NASA CMR to retrieve the data access URL for the granule and for the DMR++ file. It then retrieves the DMR++ file and injects the data URL so that data access can proceed as described above.

More on the REST Path can be found here. You need access to the NASA ESDIS Earthdata Wiki to follow that link.

Appenix A: Useful Docker Commands

A useful docker command, `ps`, provides a way to see which docker containers are running.

```
% docker ps
```

or make a command alais for a more compact listing than the default output of `docker ps`

```
% alias d-ps='docker ps --format "table {{.ID}}\t{{.Names}}\t{{.Status}}\t{{.Image}}"'
```

This will show a somewhat easier-to-read bit of information about all the running Docker containers on your host:

```
% d-ps

CCONTAINER ID   NAMES     STATUS          IMAGE
82074fe6ccfe    hyrax     Up 13 minutes   opendap/hyrax:1.17.1-126
```

If you want to stop the container, use

```
% docker rm -f hyrax
```

Appenix B: Building DMR++ files for HDF4 and HDF4-EOS2 (advanced)

This appendix documents an advanced command (`get_dmrpp_h4`) that is used to build DMR++ documents for HDF4 and HDF4-EOS2. For most people we recommend using `gen_dmrpp_side_car` instead. *Caveat emptor*. See Section 4 for up-to-date information on building the DMR++.

The HDF4 and HDF4-EOS2 (hereafter just HDF4) DMR++ document builder is currently available in the docker container we build for hyrax server/service. You can get this container from our public Docker Hub repository. You can also get and build the "Hyrax" source code and use the client that way (as part of a source code build), but it's much more complex than getting the Docker container. In addition, the Docker container includes a server that can test the DMR++ documents that are built and can even show you how the files would look when served without using the DMR++.

The following commands should be considered still experimental and subject to some change. Modify it to suit your own needs.

B.1. Using get_dmrpp_h4

Make a new directory in a convenient place and copy the HDF4 and/or HDF4-EOS2 files in that directory. Once you have the files in that directory, make an environment variable so it can be referred to easily. From inside the directory:

```
export HDF4_DIR=$(pwd)
```

Get the Docker container from Docker Hub using this command:

```
docker run -d -h hyrax -p 8080:8080 -v $HDF4_DIR:/usr/share/hyrax --name=hyrax
opendap/hyrax:snapshot
```

What the options mean:

```
-d, --detach Run container in background and print container ID
-h, --hostname Container host name
-p, --publish Publish a container's port(s) to the host
-v, --volume Bind mount a volume
--name Assign a name to the container
```

This command will fetch the container **opendap/hyrax:snapshot** from Docker Hub. The *snapshot* is the most recent build of the container. It will then *run* the container and return the container ID. The *hyrax* server is now running on your computer and can be accessed with a web browser, curl, etc. More on that in a bit.

The volume mount, from `$HDF4_DIR` to `'/usr/share/hyrax'` mounts the current directory of the host computer running the container to the directory */usr/share/hyrax* inside the container. That directory is the root of the server's data tree. This means that the HDF4 files you copied into the `HDF4_DIR` directory will be accessible by the server running in the container. That will be useful for testing later on.

Note: If you want to use a specific container version, substitute the version info for *snapshot*.

B.1.1. Running the DMR++ builder

At the end of this, I'll include a shell script that takes away many of these steps. However, the script obscures some aspects of the command that you might want to tweak, so the following shows you all the details. Skip to **Simple shell command** to skip over these details.

Make sure you are in the directory with the HDF4 files for these steps.

Get the command to return its help information:

```
docker exec -it hyrax get_dmrpp_h4 -h
```

will return:

```
usage: get_dmrpp_h4 [-h] -i I [-c CONF] [-s] [-u DATA_URL] [-D] [-v]

Build a dmrpp file for an HDF4 file. get_dmrpp_h4 -i h4_file_name. A dmrpp
file that uses the HDF4 file name will be generated.

optional arguments:

...
```

Let's build a DMR++ now, by explicitly using the container:

```
docker exec -it hyrax bash
```

starts the *bash* shell in the container, with the current directory as root (/)

```
[root@hyrax /]#
```

Change to the directory that is the root of the data (you'll see your HDF4 files in here):

```
 cd /usr/share/hyrax
```

You will see, roughly:

```
[root@hyrax /]# cd /usr/share/hyrax
[root@hyrax hyrax]# ls
3B42.19980101.00.7.HDF
3B42.19980101.03.7.HDF
3B42.19980101.06.7.HDF
...
```

In that directory, use the *get_dmrpp_h4* command to build a DMR++ document for one of the files:

```
[root@hyrax hyrax]# get_dmrpp_h4 -i 3B42.20130111.09.7.HDF -u
'file:///usr/share/hyrax/3B42.20130111.09.7.HDF'
```

Copy that pattern for whatever file you use. From the `/usr/share/hyrax` directory, you pass *get_dmrpp_h4* the name of the file (because it's local to the current directory) using the **-i** option. The **-u** option tells the command to embed the URL that follows it in the DMR++. I've used a *file://* URL to the file */usr/share/hyrax/3B42.19980101.00.7.HDF*.

In the URL above, three slashes follow the colon: two from the way a URL names a protocol and one because the pathname starts at the root directory.

Building the DMR++ and embedding a *file://* URL will enable testing the DMR++.

B.1.2. Simple shell command

Here is a simple shell command that you can run on the host computer that will eliminate most of the above.

"In the spirit of a recipe, I'll restate the earlier command for starting the docker container with the **get_dmrpp_h4** command and the **hyrax** server."

Start the container:

```
docker run -d -h hyrax -p 8080:8080 -v $HDF4_DIR:/usr/share/hyrax --name=hyrax
opendap/hyrax:snapshot
```

Check if it is running:

```
docker ps
```

The command, written for the Bourne Shell, is:

```
#!/bin/sh
```

```
#
# usage get_dmrpp_h4.sh <file>

data_root=/usr/share/hyrax

cat <<EOF | docker exec --interactive hyrax sh
cd $data_root
get_dmrpp_h4 -i $1 -u "file://$data_root/$1"
EOF
```

Copy that, save it in a file (I named the file *get_dmrpp_h4.sh*).

Run the command on the host, not the docker container, and in the directory with the HDF4 files (you don't have to do that, but sorting out the details is left as an exercise for the reader. Run the command like this:

```
./get_dmrpp_h4.sh AMSR_E_L3_SeaIce25km_V15_20020601.hdf
```

The DMR++ will appear when the command completes.

```
(hyrax500) hyrax_git/HDF4-dir % ls -l
total 1251240
-rw-r--r--@ 1 jimg  staff    1250778 Aug 22 22:31
AMSR_E_L2_Land_V09_200206191112_A.hdf
-rw-r--r--@ 1 jimg  staff   20746207 Aug 22 22:32
AMSR_E_L3_SeaIce25km_V15_20020601.hdf
-rw-r--r--  1 jimg  staff    3378674 Aug 28 17:37
AMSR_E_L3_SeaIce25km_V15_20020601.hdf.dmrpp
```

Appenix C: Building DMR++ files for HDF5/NetCDF4 with *get_dmrpp*

This appendix documents a deprecated command (`get_dmrpp`) that was used to build DMR++ documents for HDF5 and netCDF4 before `gen_dmrpp_side_car` was introduced. *Caveat emptor*. See Section 4 for up-to-date information on building the DMR++.

The application that builds the DMR++ files is a command line tool called *get_dmrpp*. It in turn uses other executables such as *build_dmrpp*, *reduce_mdf*, *merge_dmrpp* (which rely in turn on the *HDF5_handler* and the HDF5 library), along with a number of UNIX shell commands.

All of these components are installed with each recent version of the Hyrax Data Server

You can see the *get_dmrpp* usage statement with the command:

```
get_dmrpp -h
```

C.1. Using *get_dmrpp*

The way that *get_dmrpp* is invoked controls the way that the data are ultimately represented in the resulting DMR++ file(s).

The *get_dmrpp* application uses software from the Hyrax data server to produce the base DMR document which is used to construct the DMR++ file.

The Hyrax server has a long list of configuration options, several of which can substantially alter the structural and semantic representation of the dataset as seen in the DMR++ files generated using these options.

C.2. Command line options

The command line switches provide a way to control the output of the tool. In addition to common options like verbose output or testing modes, the tool provides options to build extra (aka 'sidecar') data files that hold information needed for CF compliance. See the "missing data" section for more information. In addition, it is often desirable to build DMR++ files before the source data files are uploaded to a cloud store like S3. In this case, the URL to the data may not be known when the DMR++ is built. We support this by using placeholder/template strings in the "dmr++" and which can then be replaced with the URL at runtime, when the DMR++ file is evaluated. See the '-u' and '-p' options below.

C.2.1. Inputs

-b

The fully qualified path to the top-level data directory. Data files read by *get_dmrpp* must be in the directory tree rooted at this location and their names expressed as a path relative to this location. The value may not be set to `/` , or `/etc`. The default value is `/tmp` if a value is not provided. All the data files to be processed must be in this directory or one of its subdirectories. If *get_dmrpp* is being executed from same directory as the data then `-b \`pwd\`` or `-b .` works as well.

-u

This option is used to specify the location of the binary data object. Its value must be `http`, `https`, or a `file://` URL. This URL will be injected into the DMR++ when it is constructed. If option `-u` is not used; then the template string `OPeNDAP_DMRpp_DATA_ACCESS_URL` will be used and the DMR++ will substitute a value at runtime.

-c

The path to an alternate bes configuration file to use.

-s

The path to an optional addendum configuration file which will be appended to the default BES

configuration. Much like the `site.conf` file works for the full server deployment it will be loaded last and the settings there-in will affect the default configuration.

C.2.2. Output

-o

The name of the file to create.

C.2.3. Verbose Output Modes

-h

Show the help/usage page.

-v

verbose mode, prints the intermediate DMR.

-V

Very verbose mode, prints the DMR, the command, and the configuration file used to build the DMR.

-D

Print the DMR that will be used to build the DMR++.

-X

Do not remove temporary files. May be used independently of the `-v` and/or `-V` options.

C.2.4. Tests

-T

Run ALL hyrax tests on the resulting DMR++ file and compare the responses the ones generated by the source HDF5 file.

-I

Run hyrax inventory tests on the resulting DMR++ file and compare the responses the ones generated by the source HDF5 file.

-F

Run hyrax value probe tests on the resulting DMR++ file and compare the responses the ones generated by the source HDF5 file.

C.2.5. Missing Data Creation

-M

Build a 'sidecar' file that holds missing information needed for CF compliance (e.g., Latitude, Longitude and Time coordinate data).

-p

Provide the URL for the Missing data sidecar file. If this is not given (but -M is), then a template

value is used in the DMR++ file and a real URL is substituted at runtime.

-r

The path to the file that contains missing variable information for sets of input data files that share common missing variables. The file will be created if it doesn't exist and the result may be used in later invocations of *get_dmrpp* (using `-r`) to identify the missing variable file.

C.2.6. AWS Integration

The *get_dmrpp* application supports both S3 hosted granules as inputs, and uploading generated DMR++ files to an S3 bucket.

S3 Hosted granules are supported by default

When the `get_dmrpp` application sees that the name of the input file is an S3 URL it will check to see if the AWS CLI is configured and if so `get_dmrpp` will attempt retrieve the granule and make a DMR++ utilizing whatever other options have been chosen. **For example:**

```
get_dmrpp -b `pwd` s3://bucket_name/granule_object_id
```

-U

The `-U` command line parameter for `get_dmrpp` instructs `get_dmrpp` application to upload the generated DMR++ file to S3, but only when the following conditions are met:

- The name of the input file is an S3 URL.
- The `AWS CLI` has been configured with credentials that provide `r+w` permissions for the bucket referenced in the input file S3 URL.
- The `-U` option has been specified. If all three of the above are true then `get_dmrpp` will copy the retrieve the granule, create a DMR++ file from the granule, and copy the resulting DMR++ file (as defined by the `-o` option) to the source S3 bucket using the well-known NGAP sidecar file naming convention: **s3://bucket_name/granule_object_id.dmrpp**. For example:

  ```
  get_dmrpp -U -o foo -b `pwd` s3://bucket_name/granule_object_id
  ```

C.3. *HDF5_handler* Configuration

Because *get_dmrpp* uses the *HDF5_handler* software to build the DMR++ the software must inject the *HDF5_handler*'s configuration.

The default configuration is large, but any valued may be altered at runtime.

Here are some of the commonly manipulated configuration parameters with their default values:

```
H5.EnableCF=false
H5.EnableDMR64bitInt=true
H5.DefaultHandleDimension=true
```

```
H5.KeepVarLeadingUnderscore=false
H5.EnableCheckNameClashing=true
H5.EnableAddPathAttrs=true
H5.EnableDropLongString=true
H5.DisableStructMetaAttr=true
H5.EnableFillValueCheck=true
H5.CheckIgnoreObj=false
```

C.3.1. Note to DAACs with existing Hyrax deployments.

If your group is already serving data with Hyrax and the data representations that are generated by your Hyrax server are satisfactory, then a careful inspection of the localized configuration, typically held in `/etc/bes/site.conf`, will help you determine what configuration state you may need to inject into *get_dmrpp*.

C.4. The *H5.EnableCF* option

Of particular importance is the *H5.EnableCF* option, which instructs the *get_dmrpp* tool to produce Climate Forecast convention (CF) compatible output based on metadata found in the granule file being processed.

Changing the value of *H5.EnableCF* from **false** to **true** will have (at least) two significant effects.

It will:

- Cause *get_dmrpp* to attempt to make the dmr++ metadata CF compliant.
- Remove Group hierarchies (if any) in the underlying data granule by flattening the Group hierarchy into the variable names.

By default *get_dmrpp* the *H5.EnableCF* option is set to false:

```
H5.EnableCF = false
```

There is a much more comprehensive discussion of this key feature, and others, in the HDF5 Handler section of the Appendix in the Hyrax Data Server Installation and Configuration Guide.

C.5. Missing data, the CF conventions and *HDF5*

Many of the *HDF5* files produced by NASA and others do not contain the domain coordinate data (such as latitude, longitude, time, etc.) as a collection of explicit values. Instead, information contained in the dataset metadata can be used to reproduce these values.

In order for a dataset to be Climate Forecast (CF) compatible, it must contain these domain coordinate data values.

The Hyrax *HDF5_handler* software, used by the *get_dmrpp* application, can create this data from the dataset metadata. The *get_dmrpp* application places these generated data in a “sidecar” file for

deployment with the source *HDF5/netcdf-4* file.

[1] The HTTP/S service must support the *Range* header of HTTP/1.1. When using libcurl, both HTTP/S and the 'file:' protocols can be used.

[2] The netCDF4 format is a subset of HDF5, so HDF5 tools are used for both.

[3] Not supporting netCDF3 is a shame because it's commonly found in older collections of data, and it's one of the simpler data formats.