Time Series Data Server (TSDS)

Standards-Compliant, Convenient, and Efficient Access to Time Series Data

Douglas M **Lindholm**, Robert S Weigel, Anne Wilson, Alexandria Ware DeWolfe
doug.lindholm@lasp.colorado.edu

GEORGE MASON UNIVERSITY
LASP LABORATORY FOR ATMOSPHERIC AND SPACE PHYSICS
UNIVERSITY OF COLORADO AT BOULDER

OPeNDAP

OPeNDAP (Open-source Project for a Network Data Access Protocol, http://opendap.org/) is a data transport architecture and protocol widely used by earth scientists. OPeNDAP.org provides reference implementations of servers which are commonly used by data providers. Most importantly, OPeNDAP goes beyond releasing free software, they define a specification of a Data Access Protocol (DAP 2.0) that is an accepted standard (NASA Earth Science Data Systems Recommendation ESE-RFC-004).

The **Data Access Protocol (DAP)** defines an HTTP based client-server protocol for requesting and delivering data across the Internet. The request is represented in an HTTP URL using an intuitive syntax. The server returns the requested data subset in a standard, machine-readable form. A compliant server will also serve various forms of metadata.

OPeNDAP URL Syntax:
http://host/server/dataset.suffix?constraint_expression
- *host*: Name of the computer running the server
- *server*: Name of the server (e.g. TSDS)
- *dataset*: Name of a dataset that the server can serve
- *suffix*: The type/format of the output
- *constraint_expression*: A collection of optional request parameters such as variables, range, and functions.

Some standard OPeNDAP suffixes:
- *dds*: Dataset Descriptor Structure (ASCII)
- *das*: Dataset Attribute Structure (ASCII)
- *dods*: Data as defined by the Data Access Protocol (DAP)
- *info*: Information about the dataset and request options
- *html*: HTML view of dataset information and a form for requesting data

Clients

OPeNDAP.org also provides client software and the community has contributed numerous client software libraries and applications. Clients typically take a form as represented immediately to the right.

Web Application: The DAP "html" suffix will return an order form to a browser allowing the user to request data. A more sophisticated web interface can be presented, providing a more complete user experience. The LASP Interactive Solar Irradiance Data Center (LISIRD, http://lasp.colorado.edu/lisird-beta/), for example, uses JavaScript, Flash, and Ajax to provide interactive plots to explore the data and to request subsets or aggregations of data sets.

Web Browser: A user can enter a raw DAP request in a Web browser and directly get the results.

Application Programming Interface (API): A user can write code to read data from the server directly into their program. Client APIs are available for many programming languages including IDL, Matlab, Java, and C/C++.

Applications: Many third party scientific visualization and analysis applications include support for accessing data via OPeNDAP (http://opendap.org/faq/whatClients.html). The TSDS includes a comma separated value (csv) output option that can be read into a spreadsheet application.

Overview

The Time Series Data Server (TSDS) is a software framework that enables data providers to easily and efficiently serve data that are a function of time. It implements the RESTful (i.e. simple HTTP request and response) OPeNDAP web service interface as defined by the open standard Data Access Protocol (DAP). Because the TSDS interface is based on open standards, many clients are already available for users to access subsets and aggregations of the data it serves. Because it makes the simplifying assumption that it serves time series data, it can bypass many of the complications that other OPeNDAP servers have to deal with and it can optimize for efficient access to time subsets regardless of the data set size.

The TSDS uses the open standards-based NetCDF-Java API and NetCDF Markup Language (NcML) files, and in some cases custom adapters, to read subsets and aggregations of data sets in their native format from a variety of sources. Data sets are represented in terms of the Common Data Model (CDM) internally. Given the ability to read many data sets into the CDM, a function interface to manipulate the data on the server, and a variety of output format options, the TSDS can be configured by the data provider to support many client requested operations.

The TSDS has an object-oriented architecture implemented using Java Servlet technology. Installation simply involves dropping its Web archive (war) file into a Servlet container (such as Tomcat). Its modular design enables a TSDS installation to be easily configured and expanded. Plug-able server-side functions and writers can be added to the system by simply including an entry in a configuration file mapping function names to Function implementations and mapping output types to Writer implementations. The Function and Writer interfaces can easily be implemented to provide additional functionality.

The TSDS software is Open Source and will soon be publicly available. The code is designed for ease of use. Most new functionality can be implemented as a plug-in without affecting any other code. An ant build script makes it trivial to compile and build a web archive (war) file. The TSDS is expected to evolve with user-driven and community-provided enhancements. The version 1.0 release will be production ready. A 2.0 development version will allow the API to evolve in response to interactions with the NetCDF, OPeNDAP, and data provider communities.

NetCDF

NetCDF (Network Common Data Form, http://www.unidata.ucar.edu/software/netcdf/) is widely known as a self-describing, machine-independent, open standard scientific data file format. The Unidata Program Center at the University Corporation for Atmospheric Research (UCAR) created NetCDF and continues to develop software and tools for reading and writing NetCDF files.

NetCDF is more that just a data format, it defines the **Common Data Model** (**CDM**, http://www.unidata.ucar.edu/software/netcdf-java/CDM/) which merges the data models of NetCDF, OPeNDAP, and HDF5 (Hierarchical Data Format, http://www.hdfgroup.org/HDF5/). The CDM describes the logical structure of scientific data sets. It includes components such as Dataset, Group, Variable, Dimension, and Attribute. TSDS uses the NetCDF-Java library which implements the CDM.

NetCDF Markup Language (**NcML**, http://www.unidata.ucar.edu/software/netcdf/ncml/) is an XML representation of NetCDF metadata. The NetCDF software can read data from a CDM data set via a NcML file that describes it (as exemplified immediately to the left). NcML can expose a subset of a data set or an aggregation of multiple data sources. A single NcML file can be used as the access point to a virtual data set.

Additional data sets can be adapted for reading by the NetCDF-Java API by defining the data set in terms of the Common Data Model in an NcML file and providing an implementation of NetCDF's **IOServiceProvider** interface. This can be a rather straight forward way to provide data in its native format, especially for the simple structure of time series data.

Data Sources

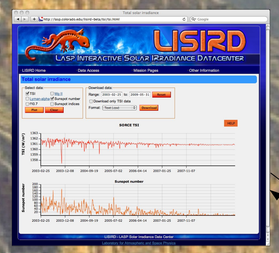
The NetCDF-Java API comes with support for reading NetCDF, OPeNDAP, HDF5, and a few other formats exposed in terms of the Common Data Model via an IOServiceProvider implementation. The TSDS provides additional IOServiceProvider implementations for tabulated ASCII, flat binary, and relational database data sources. NcML can be used to enhance the metadata, subset, and/or aggregate datasets of these forms. Work is also being done to serve data from other web services, including a separate TSDS instance. Given the modular, open nature of the TSDS architecture, there's no limit to what the scientific data community might come up with.

The Time Series Data Server starts with the simplifying assumption that all the data it serves can be represented by a Sequence with time as the independent variable mapped to one or more data samples. A data sample can take the form of a CDM variable type: a Scalar (e.g. temperature), a Structure representing a logical grouping of variables (e.g. magnetic field components), or a Sequence representing a functional relationship between an independent variable and one or more dependent variables (e.g. spectrum).

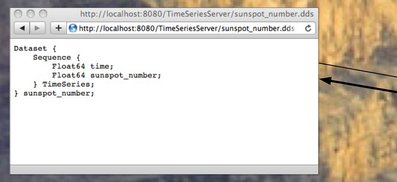
The Common Data Model also defines a Grid type. Future TSDS releases may support serving image, map, and 3D geospatial data types. There is no reason why a TSDS service couldn't be used in coordination with another service (e.g. coordinate system transform) to create a new aggregated service. However, the goal of the Time Series Data Server will always be to provide efficient access to subsets of a time series, regardless of the data types on the dependent side of that function.

References
TSDS on SourceForge: http://tsds.sourceforge.net/
OPeNDAP: http://opendap.org/
OPeNDAP on Wikipedia: http://en.wikipedia.org/wiki/OPeNDAP
DAP 2.0 Specif cation: http://www.esdswg.org/spg/rfc/ese-rfc-004
NetCDF: http://www.unidata.ucar.edu/software/netcdf-java/
NetCDF on Wikipedia: http://en.wikipedia.org/wiki/NetCDF
NcML: http://www.unidata.ucar.edu/software/netcdf/ncml/
CDM: http://www.unidata.ucar.edu/software/netcdf-java/CDM/

Clients



Web Applications

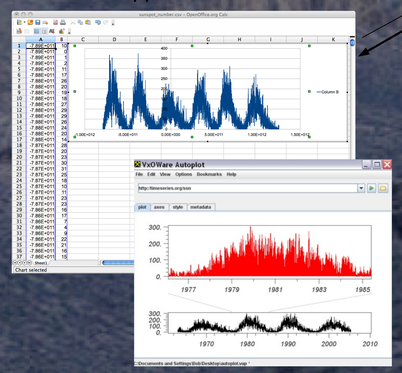


Direct Web Browser Access

Software API

```
data = get_data(dataset, start, stop)
time = data.time
ssn = data.sunspot_number
plot, time, ssn
```

Applications



Time Series Data Server

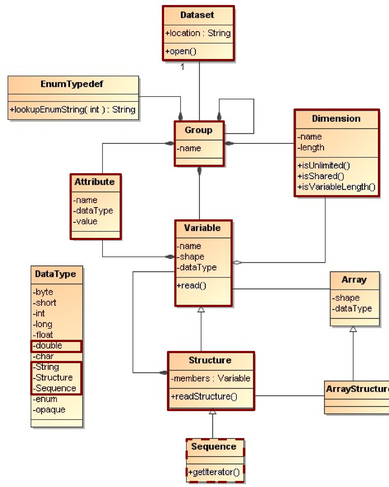
OpenDAP Java Servlet

dods | dds | das | info | html | csv | nc | sav | pro | ...

thin | convert_units | replace_missing | ...

Common Data Model

http://www.unidata.ucar.edu/software/netcdf-java/CDM/



Components outlined in **red** are currently implemented.

NetCDF Java API

ASCII | binary | Database | ...

Data Sources

Remote file

NcML

NcML for Remote File

```
<netcdf location="http://example.org/ssn.nc">
  <dimension name="time"/>
  <variable name="time"
            shape="time"
            type="double">
    <attribute name="units"
      value="seconds since 1970-01-01" />
  </variable>
  <variable name="sunspot_number"
            shape="time"
            type="double">
    <attribute name="missing_value"
      type="double" value="-999" />
  </variable>
</netcdf>
```

Data collection

NcML to aggregate a collection of time stamped HDF5 files

```
<netcdf>
  <aggregation dimName="time"
               type="joinNew">
    <scan location="/data/"
      dateFormatMark="ssn#yyyy"
      suffix=".h5"/>
  </aggregation>
</netcdf>
```

ASCII data table

NcML for a local ASCII file with a custom IOServiceProvider with internally defined times with a 2D vector variable

```
<netcdf location="data/field.txt"
  iosp="tsds.iosp.AsciiOSP">
  <dimension name="time"/>
  <variable name="time" length="1000"
            shape="time" type="double">
    <attribute name="units"
      value="days since 1985-01-01" />
    <values start="0" increment="1" />
  </variable>
  <variable name="B" type="Structure"
            shape="time">
    <variable name="x" type="double" />
    <variable name="y" type="double" />
  </variable>
</netcdf>
```

Raw data

NcML to aggregate variables from two data sources

```
<netcdf>
  <aggregation type="union">
    <netcdf location="data/time.bin"
      iosp="tsds.iosp.BinIOSP">
      <dimension name="time"/>
      <variable name="time"
                shape="time" type="double">
        <attribute name="units"
          value="days since 1945-01-01"/>
      </variable>
    </netcdf>
    <netcdf iosp="tsds.iosp.DatabaseIOSP"
      iospParam="jdbc:connection URL">
      <dimension name="time"/>
      <variable name="sunspot_number"
                shape="time" type="double">
      </variable>
    </netcdf>
  </aggregation>
</netcdf>
```

Relational Database

NcML for an imagined service interface to a VxO

```
<netcdf iosp="my.iosp.MyVxOIOSP"
  iospParam="myvxo query('sunspot')">
</netcdf>
```

Virtual Observatory web service

IOServiceProvider

```
# Define the dataset in terms of the CDM
open(file, netcdfFile)
# Read subset of the given variable
readData(variable, section)
```

my.AsciiWriter

writeDataset()

my.MovingAverage

applyFunction()

```
# Writer properties
# Define the Writer subclass and properties
# to use for a given request “suffix”.

# Invoke with “csv” suffix in the request
writer.csv.class = my.AsciiWriter
writer.csv.format = %8.2f
writer.csv.delimeter = ,
```

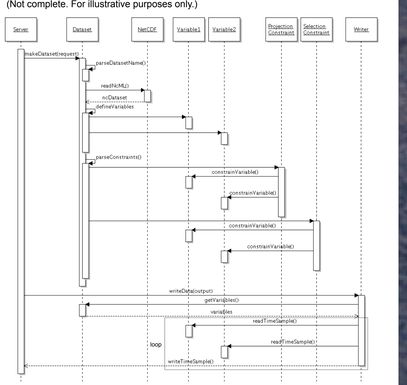
```
# Function properties
# Define the Function subclass and properties
# to apply for a requested function call.

# Invoke with “smooth()” in the request
function.smooth.class = my.MovingAverage
function.smooth.width = 5
```

Property Files

Custom Implementations

UML Diagram of the TSDS Data Request Sequence

(Not complete. For illustrative purposes only.)



TSDS Class Diagram

(Not complete. For illustrative purposes only.)