
# Version 2.2.2 Interface Reference

*Revision 2017-05-22*

| Purpose | Who | Date |
|---|---|---|
| Starting Das 2.2.1 features, server side caching directives | C. Piker | 2015-03-16 |
| Adding exmapleParam to DSDFs | C. Piker | 2015-05-12 |
| Added information on macro substitution in label strings | C. Piker | 2015-05-21 |
| Added keyword to handle dataset path redirects | C. Piker | 2015-06-29 |
| Expand validRange DSDF info, added item_XX description | C. Piker | 2015-07-15 |
| Added yTagInterval, yTagMin, yTagMax, corrections, clarifications | C. Piker | 2017-05-08 |
| | | |
| | | |
| | | |

*Recent Revisions*

Dept. Of Physics & Astronomy
The University of Iowa
Iowa City, IA 52242

# Table of Contents

| Acronym | Meaning |
|---|---|
| DSDF | Data Set Descriptor File, A file defining a Das 2.1 data source for use via a Das2 server. |
| DSID | Data Set IDentification, A type of file defining a Das 2.2 data source for use via a Das2 server. |
| HTTP | HyperText Transfer Protocol, The basic transport protocol for most of the world's data. |
| IDL | Interactive Data Language, a programming language that runs on interpreters written by the Exelis company. |
| | |
| | |
| | |

*Table 1: Acronyms used within this document*

| Issue | Affects |
|---|---|
| | |
| | |
| | |

*Table 2: Known document Issues*

# 1 Das2 System

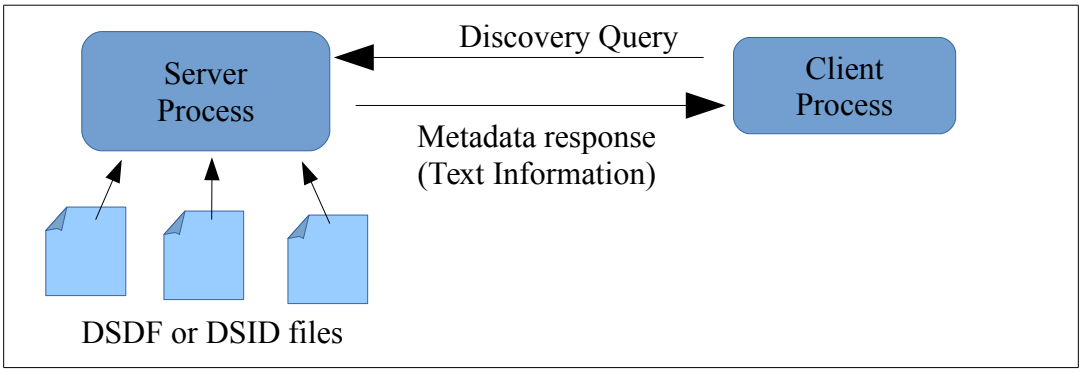
This interface control document defines the interaction protocols and conventions for the Das2 data handling system. Das2, and it's predecessor Das1, were developed at the University of Iowa to ease retrieval and use of space physics research data for Professor Don Gurnett's staff and collaborators. Data from the Voyager, Galileo, Cassini, and Juno spacecraft are routinely accessed and processed via Das2 servers. In 2012 Professor Craig Kletzing's staff also began using a Das2 server in support of the Van Allen Probes mission. As more programs and missions come to rely on the system the time has come to document the interfaces by which all Das2 developers may cooperate.

## 1.1 Overview

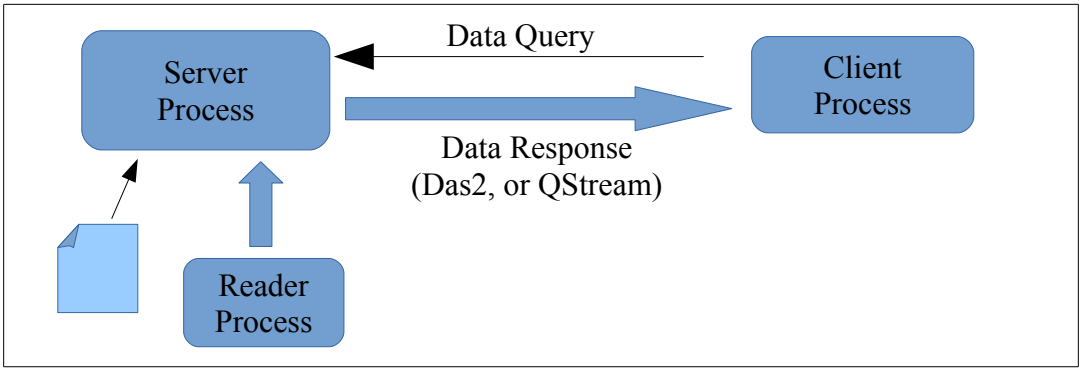
The Das2 system consists of three basic parts, in order of further removal from the source data

1. One or more **readers**. These programs read data from some source and provide it in a standardize format. Readers are accompanied via Data Source Identification files (DSID files) which describe how to run the reader, and what kind of data it provides. There is also an older data source definition file format, called DSDF format, that consists of IDL statements.
2. One or more **servers**. These programs listen for queries and respond. If the query is a request for data it starts a reader and provides the data.
3. One or more **clients**. These programs are the human interface. They request data and plot it.

Clients may request general information about the server, or one of more of it's data sources. These are termed discovery queries. Currently only the HTTP protocol is supported as the transport mechanism for discovery queries. The basic information flow is diagrammed below.




*Discovery Query Information Flow*




*Data Query Information Flow*

Clients may also request data from one or more of the server's data sources. To handle a data request, a server

starts a reader program, provides the reader with a set of query parameters, and pipes it output through any needed server side filters and then sends the final output across the network to the client program. Regardless of the on-disk format of the original data, the data response is encoded as either a Das2 Stream or a QStream.

## 1.2 Known Das2 Servers

It's handy to be able to test communications with a Das2 server when using the system reference. Towards that end, Table 3 provides a list of known Das2 Servers as of summer 2013. The authors make no guarantee on it's continued accuracy.

| Hostname | Resource | Protocol | Purpose |
|---|---|---|---|
| `www-pw.physics.uiowa.edu` | `/das/das2Server` | Das 2.1 | Serves data for instruments on Cluster, Galileo, Polar and others |
| `emfisis.physics.uiowa.edu` | `/das/das2Server` | Das 2.1 | Serves data for instruments on the Van Allen Probes mission |
| `planet.physics.uiowa.edu` | `/das/das2Server` | Das 2.1 | Serves data for Juno, Voyager and others |
| `cassini.physics.uiowa.edu` | `/das/das2Server` | Das 2.1 | Serves data for instruments on Cassini |

*Table 3: Known Das2 Server Locations*

## 1.3 Autoplot – A General Das2 Client

For basic testing a decades old telnet client may be used to interact with a Das2 server. But this is not very satisfying and hardly suitable for getting real work done. The Autoplot program provides an interface for browsing Das2 servers, retrieving data, and interacting with the resulting plots. A copy of Autoplot may be downloaded from:

http://autoplot.org

## 1.4 Data Formats

Currently two transport formats are supported. All three are considered stream formats in that client programs may begin processing data before the transmission completes. Data definitions always proceed data values.

**das2Streams** – These are self-describing data sets which contain headers to define the layout of succeeding bytes. This format supports a handful of fixed data value arrangements.

**QStreams** – These are also self-describing data sets which contain both headers to define the layout of succeeding bytes. This format is more general than das2Streams, supporting a wider variety of data value arrangements.

## 1.5 What about Das1?

The predecessor to Das2 was just named Das, though with the creation of a successor it has taken on the name Das1. The Das1 system defined data sources via files containing snippets of IDL code and the data sources themselves were to output a stream consisting entirely of Big Endian IEEE 32-bit floating point numbers. These streams contained no headers or other descriptive information and were thus not complete without the IDL code fragment defining their structure. Furthermore there was no standard method by which Das1 data sets could be transmitted over the network to a remote client program. Typically these data sets were rendered into image files on a server, and only the static images themselves were transported to the end user program.

Das2 replaced Das1 by providing fully interactive client programs which could manipulate data, and automatically issue new data queries as a human navigated the data set. Das2 servers can make use of Das1 reader programs though continued development of Das1 readers is not recommended.

# 2 Das 2.2 Client – Server Interface

The purpose of a Das2 server is to listen for queries and provide responses in a standardized format. Though there are many moving parts behind the scenes, client programs need only implement a Client – Server Interface in order to find and retrieve data. If programs need to actually parse and display data sections 4 and 5, will be needed as well.

The Das 2 client - server protocol has been use since 2003. In this protocol **all** data are assumed to be indexed by collection a single parameter (almost always the parameter is Time), and **all** queries that return a data response require a starting and ending point.

Data discovery consists of HTTP GET requests sent by the client program, followed by an HTTP response whose message body that contains the response data.[1] To nail it down, requests have the following syntax:

**Generic Das 2 Request**

```
GET RESOURCE?QUERYSTRING HTTP/1.0
Host: SERVER
Accept: text/*
```

Note that each line is terminated by the newline character (ASCII 0x0A). The items in bold are replacement parameters.

**NOTE:** All data requests are HTTP message headers and thus *end with a single blank line*.

## 2.1 List Query

This request provides a list of all legacy data sources defined using DSDF format files. The following example text, if encoded as ASCII bytes, could be sent to a server. Here's an example of a list discovery request.

**List Request**

```
GET /das2/das2Server?server=list HTTP/1.0
Host: www-pw.physics.uiowa.edu
Accept: *
```

Which would return one line for each level of the data source hierarchy and one line for each data source similar to the following (only the first few return lines are given):

**List Response**

```
HTTP/1.1 200 OK
Date: Tue, 23 Jul 2013 21:04:06 GMT
Server: Apache/2.0.63
Connection: close
Content-Type: text/plain; charset=ISO-8859-1

cassini/
cassini/mag/
cassini/mag/mag_vector
cassini/mag/mag_vectorP
cassini/overlay/
```

*(More lines follow...)*

Each level of the data source hierarchy is returned, followed by each data source within that level of the hierarchy.

1 *HTTP The Definitive Guide*, Chapter 3, O'Reilly & Associates, Inc, 2002

Note that hierarchy levels end in a trailing '/' character.

## 2.2 Discovery Query

This request is similar to a **`list`** request but only returns data sources that have an example time range.

**Discovery Request**

```
GET /das2/das2Server?server=discovery HTTP/1.0
Host: www-pw.physics.uiowa.edu
Accept: *
```

Since the `cassini/` level below contains no data sources that have an example time range, that level of the hierarchy isn't returned.

**Discovery Response**

```
HTTP/1.1 200 OK
Date: Tue, 23 Jul 2013 21:10:53 GMT
Server: Apache/2.0.63
Connection: close
Content-Type: text/plain; charset=ISO-8859-1

cassini/mag/
cassini/mag/mag_vector
cassini/mag/mag_vectorP
das2_1/testing/
```

*(More lines follow...)*

## 2.3 DSFD Query

This request provides information on a particular data source, thus two parameters are required. In the following example a more information is requested on magnetic field vector data set:

**DSDF Request**

```
GET /das2/das2Server?server=dsdf&dataset=cassini/mag/mag_vector HTTP/1.0
Host: www-pw.physics.uiowa.edu
Accept: *
```

This returns a Das2 Stream packet providing some information on the data set, it this case:

**DSDF Response**

```
HTTP/1.1 200 OK
Date: Tue, 23 Jul 2013 21:21:55 GMT
Server: Apache/2.0.63
Expires: Tue, 23 Jul 2013 21:21:55 GMT
Connection: close
Content-Type: text/plain; charset=ISO-8859-1

[00]000176<stream > <properties validRange="1999-228 to 2010-359"
server="http://cassini.physics.uiowa.edu/das/das2Server" das2Stream="1"
exampleRange="2010-001 to 2010-002" /> </stream>
```

is returned. Arbitrary identifiers may be returned as attributes in the `properties` element. There are no particular rules just whatever happens to be in the server side DSDF file is sent over. Some conventions have been followed, but a client program can't expect this to always be the case. The conventional attributes are:

**das2stream= "0"|"1"** If 1 the data source probably provides data in Das2 Stream format.

**qstream= "0"|"1"** If 1 the data source probably provides data in QStream format.

**exampleRange= "*START_TIMEPOINT* to *END_TIMEPOINT*"** If returned, this attribute value probably contains a query range that will provide some data.

**server= "*URL*"** If returned, this attribute's value contains the proper Das2 Server to query to retrieve data for this data source. Das2 Servers can advertise other server's data sources.

Das 2.2 data discovery protocol messages are more tightly specified.

## 2.4 Data Query

The fundamental assumption of all Das 2.1 compatible data sources is that data are queried by time range. Though still relatively simple, data requests are the most complex queries supported by the Das 2.1 server as many optional parameters are supported. The example query below uses the minimum number of data request parameters.

**Data Request**

```
GET /das/das2Server?
server=dataset&dataset=cassini/mag/mag_vector&start_time=2010-001&end_time=2010-002 HTTP/1.0
Host: cassini.physics.uiowa.edu
Accept: *
```

For this particular data source, the response body contains a Das2 Stream. For more details on the Das2 Stream format, see section 4.

**Data Response (Das2 Stream Data Source)**

```
HTTP/1.1 200 Binary data follows
Date: Tue, 23 Jul 2013 22:14:13 GMT
Server: Apache/1.3.9 (Unix)
Content-disposition: inline;filename=mag_vector_2010-001_2010-002.d2s
Expires: Tue, 23 Jul 2013 22:14:13 GMT
Connection: close
Content-Type: application/vnd.das2.das2stream

[00]000124<stream version="2.2">
  <properties start="2010-001" end="2010-002" Datum:xTagWidth="120 s" int:taskSize="100"/>
</stream>
[01]000224<packet>
  <x type="sun_real8" units="t2000"/>
  <y type="sun_real4" name="" units=""/>
  <y type="sun_real4" name="x" units=""/>
  <y type="sun_real4" name="y" units=""/>
  <y type="sun_real4" name="z" units=""/>
</packet>
:01:
```

*(Binary Data Follows...)*

Table 4 below describes the query string elements for a Das 2.1 data request.

| Key | Value | Required |
|---|---|---|
| `server` | `dataset` | yes |
| `dataset` | The data source to query | yes |
| `start_time` *(reader.x.min)* | Any parse-able time point format. The ISO-8601 time format is a subset of the allowed time formats. | yes |
| `end_time` *(reader.x.max)* | A time point greater than the value of `start_time`. | yes |
| `interval` *(reader.x.interval)* | The interval in seconds between data points. Some data sources, such as those that return spacecraft ephemerides, require this parameter, though this is not advertised. | maybe |
| `resolution` *(reducer.x.resolution)* | The time resolution in seconds at which data are needed. When the resolution is specified server side data reduction may be triggered which can drastically reduce network overhead. | no |
| `params` *(reader.params)* | Extra information may be passed to a reader via this parameter. | maybe |
| `compress` | If present and set to the value `true`, and the data source provides Das2 Stream formatted output, then the stream is g-zipped on the server before transmission over the wire. | no |
| `ascii` | If present, and set to any value, and the data source provides Das2 Stream formatted output, then binary data values will be converted to their ASCII equivalents before transmission. | no |

*Table 4: Das 2.2 Data Request Query Parameters*

It is possible for one Das2 server to advertise data sources that are hosted by a second Das2 server. If a client program queries for data from non-local data source an HTTP redirect is issued instead of a data reply. In the example below, the server `www-pw.physics.uiowa.edu` is requested to provide Cassini Mag vector data. Since the server knows that those data are provided by the host planet `.physics.uiowa.edu`, A redirect response is returned.

**Redirect Response**

```
HTTP/1.1 302 Found
Date: Thu, 25 Jul 2013 21:53:03 GMT
Server: Apache/2.0.63
Location: http://planet.physics.uiowa.edu/das/das2Server?server=dataset;dataset=cassini
%2Fmag%2Fmag_vector;start_time=2010-001;end_time=2010-002
Content-Length: 0
Connection: close
Content-Type: text/plain
```

Note that the replacement GET string in the redirect response is URL encoded, thus "`/`" characters appear as the escape sequence `%2F`.

## 2.5 Restricted Data Query

Some data sources require that a user name and password are provided as part of the data request. In these cases the server will expect that the HTTP header `Authorization` is present in the data request. The contents of this header will be inspected by the server and compared against it's internal database before data are sent. If this header is not present a "`401 Authorization Required`" status will be returned to the client instead of the requested data. The following query and response flow illustrates querying for restricted data.

**Data Request**

```
GET /das/das2Server?
server=dataset&dataset=cassini/mag/mag_vector&start_time=2010-001&end_time=2010-002 HTTP/1.0
Host: cassini.physics.uiowa.edu
Accept: *
```

**Authorization Required Response**

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Basic realm="Das2 Server"
```

At this point the client program gathers a user name and password by what ever means it prefers and tries again.

**Data Request**

```
GET /das/das2Server?
server=dataset&dataset=cassini/mag/mag_vector&start_time=2010-001&end_time=2010-002 HTTP/1.0
Host: cassini.physics.uiowa.edu
Authorization: Basic YnJpYW4tdG90dHk6T3ch
Accept: *
```

**Data Response (Das2 Stream Data Source)**

```
HTTP/1.1 200 Binary data follows
Date: Tue, 23 Jul 2013 22:14:13 GMT
Server: Apache/1.3.9 (Unix)
Content-disposition: inline;filename=mag_vector_2010-001_2010-002.d2s
Expires: Tue, 23 Jul 2013 22:14:13 GMT
Connection: close
Content-Type: application/vnd.das2.das2stream

[00]000124<stream version="2.2">
  <properties start="2010-001" end="2010-002" Datum:xTagWidth="120 s" int:taskSize="100"/>
</stream>
[01]000224<packet>
  <x type="sun_real8" units="t2000"/>
  <y type="sun_real4" name="" units=""/>
  <y type="sun_real4" name="x" units=""/>
  <y type="sun_real4" name="y" units=""/>
  <y type="sun_real4" name="z" units=""/>
</packet>
:01:
```

*(Binary Data Follows...)*

## 2.6 Peers Query

Das2 Servers can advertise the existence of other Das2 Servers, in fact a list of know servers is maintained by the server:

`http://www-pw.physics.uiowa.edu/das/das2Server`

A client program my connect this address and gather a list of known Das2 servers using a `peers` query as demonstrated below.

**Peers Query**

```
GET /das/das2Server?server=peers HTTP/1.0
Host: cassini.physics.uiowa.edu
Accept: *
```

**Peers Response**

```
HTTP/1.1 200 OK
Date: Thu, 25 Jul 2013 21:35:12 GMT
Server: Apache/1.3.9 (Unix)
Connection: close
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/xsl" href="/~eew/das2Server.xsl"?>
<das2server>
  <peers>
    <server>
      <name>www-pw</name>
      <url>http://www-pw.physics.uiowa.edu/das/das2Server</url>
      <description>Original das2 Server</description>
    </server>
    <server>
      <name>planet</name>
        <url>http://planet.physics.uiowa.edu/das/das2Server</url>
        <description>Linux das2 server</description>
    </server>
    <server>
      <name>emfisis</name>
      <url>http://emfisis.physics.uiowa.edu/das/das2Server</url>
      <description>Dan's das2 server</description>
    </server>
  </peers>
</das2server>
```

# 3 Das 2.2 Server – Reader Interface

Readers are a server side components responsible for gathering data from the input source, typically files on disk, and transmitting a stream to standard output. Readers are independent programs, thus they can be written in any language executable by the host operating system.

## 3.1 Reader Program Command Line Interface

Das 2.2 readers are always stand alone programs which must support one of the following command line patterns.

- General data readers: *program* START_TIME END_TIME EXTRA_PARAMETERS
- Variable resolution readers: *program* START_TIME END_TIME RESOLUTION_IN_SEC EXTRA_PARAM

Here the string *program* is flexible. It can be just the name of a program, or a program and it's space separated arguments. The only requirement in that shell pipes and other IO redirection operators my not be part of the program string.

The *EXTRA_PARAMETERS* passed to a program may be specified as multiple arguments or they may arrive as a single quoted argument with space separated sub-components. There are few formatting rules on the extra arguments, shell pipes and other IO operators are not permitted but that's about it.

The output of a reader program is always a Das2 Stream or a QStream.

## 3.2 Data Source Definition Files (DSDF)

In order to server data, the Das 2.2 server depends on small text files called *Data Source Definition Files* (DSDF). The location of these files depends on your particular Das2 installation. The file names must end in the suffix:

***.dsdf***

or the server will ignore them. Data source definition files use IDL syntax, thus strings are marked by single quotes ('), a semicolon ';' represents the start of a comment, and comments run to the end of the line. Each non-comment line in the must follow the

```
keyword = value
```

pattern. Both the Das2 server and end-user client programs can access DSDF files, though typically any particular value is likely to be more useful to either the server or to the clients. A summary of keywords is provided below.

| Keyword | Required | Notes |
|---|---|---|
| **reader** | yes | An arbitrary string that provides the beginning portion of the command line needed no invoke the reader, START_TIME, END_TIME and any extra arguments will be added to any arguments embedded in the string. |
| **description** | yes | A human readable description of the data source. |
| **techContact** | yes | An email address indicating who to contact if there is a problem with a reader. An example value:<br>'Some Person <some-person@uiowa.edu>' |
| **requiresInterval** | maybe | Set to '1' to indicate that the client program should specify the interval between measurements. For time based queries the interval is assumed to have units of seconds. |

| Keyword | Required | Notes |
|---|---|---|
| **das2Stream** | **yes** (unless qstream) | If set to '1' then the reader produces a Das2 Stream (see section 4 for Das2 Stream formatting information) . There is no need to include this keyword if the reader produces a Q Stream. |
| **exampleRange_XX** | **yes** | An example time that is known to provide visible data. This is use by client programs to 'just get something on the screen'. There can be up to 100 example ranges. The format of this string field is as follows:<br>`'START_TIME to END_TIME \| NAME'`<br>using Voyager as an example:<br>exampleRange_01 = '1979-03-01 to 1979-04-14\|Jupiter Encounter'<br>exampleRange_02 = '1980-11-10 to 1980-11-14\|Saturn Encounter'<br>The `'\|Name'` portion of the string is optional, but encouraged. |
| **qstream** | **yes** (unless Das2 stream) | If set to '1' the reader produces Q Streams (see section 5 for Q Stream formatting information). There is no need to include this keyword if the reader produces a Das2 Stream. |
| **techContact** | **yes** | Provides contact information for the data set incase there are technical problems. |
| **cacheLevel_XX** | no | Some Das2 servers can store pre-reduced datasets. To direct the server to generate a set of pre-reduced datasets use this keyword. The value format is:<br>`RESOLUTION \| STORAGE_PERIOD [\| PARAMS]`<br>Each resolution level of the cache is separated by '\|' (pipe) characters. A semicolon is placed between the resolution value and file duration value.<br>● RESOLUTION is a Das2 datum string, typically given in seconds, or the keyword '`intrinsic`' to indicate caching the highest resolution available from the data source.<br>● FILE_PERIOD determines the coverage period of each cache file. For RESOLUTION Datums that have units of time, the following keywords may be used: **`hourly`**, **`daily`**, **`monthly`**. The format for caching based on data indexes other than time have not been established.<br>● PARAMS A string of optional extra parameters to send to the reader when producing the cache.<br>Example **`cacheLevel_XX`** values:<br>`cacheLevel_00 = 'intrinsic \| hourly'`<br>`cacheLevel_01 = '1 s \| daily'`<br>`cacheLevel_02 = '60 s \| daily \| -r no-spikes'`<br>Cached datasets are generated via the reduction program specified in the **`reducer`** keyword thus if reducer = 'not_reducable' only 'intrinsic' resolution may be used when specifying cache levels. If any PARAMeters are specified, these are provided to the reader with underscores converted to spaces. |
| **cacheReader** | no | If this item is blank the default cacheReader for the output type (`das2Stream` or `qstream`) will be invoked to read the data cache |
| **exampleInterval_XX** | maybe | This keyword is required for each `exampleRange` when the reader requires an interval parameter. |
| **exampleInterval** | no | A synonym for `exampleInterval_00`. |

| Keyword | Required | Notes |
|---|---|---|
| **exampleParams_XX** | no | To specify a parameter set to provide to a reader when called for a given `exampleRange` use this keyword. This keyword is adds a reader parameter string to an associated `exampleRange_XX` keyword. Where the characters XX are replaced by the example number. |
| **exampleParams** | no | A synonym for `exampleParams_00`. |
| **exampleRange** | no | A synonym for `exampleRange_00`. |
| **item_XX** | no (recommended) | Used to name independent datasets in a stream. This allows a Das2 client to only load the individual planes of a Das2 stream, or individual datasets in a qdataset bundle. The format is:<br><br>`item_XX = 'ID [\| DESCRIPTION]'`<br><br>In the following example the Das2 reader that produces magnetic X, Y, Z and Magnitude values each as a separate <y> vector.<br><br>`item_00 = 'mag \| Magnetic Field Magnitude'`<br>`item_01 = 'x_gsm \| X component in the GSM frame'`<br>`item_02 = 'y_gsm \| Y component in the GSM frame'`<br>`item_03 = 'z_gsm \| Z component in the GSM frame'`<br><br>For Das2 Streams, `ID` values above must match the **`name`** attribute value given in the `<y>`, `<yscan>` and `<z>` elements in the packet headers. See section 4.3 for more information an Das2 stream packet headers.<br><br>For QStreams, `ID` values above match the **`id`** attribute in `<qdataset>` elements in the packet headers. See section 5.2 for more information on QStream packet headers.<br><br>Though this keyword is not required, it is strongly recommended for streams that produce more than one top-level dataset by default. |
| **readAccess** | no | This value controls when authorization is needed and who is authorized. The format of this field is:<br>`'`*`AUTH_METHOD`*`:`*`TEST`* `[\| `*`AUTH_METHOD`*`:`*`TEST`* `]'`<br>here *`AUTH_METHOD`* is a token stating the authorization method and *`TEST`* is the value to test against. `AUTH_METHOD` may be one of `AGE`, `GROUP` or `USER`. If **any** one authorization method succeeds then access is granted. Thus authorization methods are combined using a logical OR test.<br><br>An example for voyager follows:<br>**`readAccess =`** `'AGE:1y6m\|GROUP:voyager\|USER:don'`<br>In this example data older that 18 months is automatically authorized, so is anyone in the group `voyager`, as well as the user `don`. |
| **securityRealm** | no | This string should be provided by a Das2 client to end-user that is providing the authentication token and provides a context-clue as to what password is expected. For example:<br><br>**`securityRealm`** = 'Juno Magnetospheric Working Group'<br><br>indicates that users in the MWG should have access to these associated data stream. |

| Keyword | Required | Notes |
|---|---|---|
| **reducer** | no | One of the key benefits of serving data via a program instead of directly transmitting files is that data reduction can take place on the server which can drastically reduce the network bandwidth required to produce a plot. There are three categories to the values for this keyword.<br>● If this keyword is not specified a default data-reduction program will process the reader's output stream. By default data are averaged in the streaming dimension (see sections 4 and 5 for a definition of the streaming dimension).<br>● If set to the value "`not_reducible`" no server-side data reduction will be preformed.<br>● If set to the name of a program on the server then that program will be used to reduce data in the streaming dimension before delivery. |
| **rename** | no | If a datasource is renamed this keyword may be used in the old copy of the datasource to point clients to the new location. DSDF files containing the keyword rename are included in the output of list and discovery queries. See section 2.1 for more information on list queries. |
| **sciContact** | no | An email address providing contact information for questions about scientific usefulness or interpretation of the data source. |
| **server** | no | A das2 server can advertise the existence of data sets hosted via another das2 server. If this keyword is present and it's value doesn't match the URL of the server which was contacted, then the contacted server issues an HTTP redirect will to the client. |
| **testInterval** | maybe | This keyword is required if `testRange` is given and the reader requires an interval parameter. |
| **testRange** | no | A test time that is expected to provide unchanging visible data. It is use by automated end-to-end testing software to make sure that the output of a reader doesn't change over time. |
| **validRange** | no (recommended) | A time range over which the data source may produce output. The syntax is:<br>`validRange = BEGIN to END`<br>Instead of a timestamp, the END may be denoted by the case-insensitive word '`now`' for on-going missions. The following example is for Voyager 1 PWS Waveform data:<br>`validRange = '1977-09-05T00:00 to NOW'`<br>Though not required, usage of this keyword is highly recommended. |
| *AnyKeyword* | no | Any other key = value pair may be included in the file but only the ones above have specific uses. |

*Table 5: DSDF Keywords*

## 3.3 Example DSDF files

The following file Voyager PWS file tells the Das2 server how to run the data reader, which reducer should be used to handle long time ranges and provides a few example times to get to interesting data quicker. The required keywords are in bold font.

```
das2Stream = 1
server = 'http://planet.physics.uiowa.edu/das/das2Server'
description = 'Electric Field averages and peaks from the Voyager 1 PWS SA'
```

```
reader = '/opt/project/voyager/bin/centos5.x86_64/vgpw_sa_rdr 1'
reducer = 'peakAverageSeconds'
readAccess = 'age:1y6m|group:voyager'
validRange = '1977-09-05 to 2014-09-01'
testRange = '2009-01-01 to 2009-04-01|Regression Test Data'
exampleRange = '2014-08-31 to 2014-09-01|Latest Data'
exampleRange_01 = '1979-03-01 to 1979-04-14|Jupiter Encounter'
exampleRange_02 = '1980-11-10 to 1980-11-14|Saturn Encounter'
techContact = 'Jane Doe <jane-doe@uiowa.edu>'
sciContact = 'Dr. Scientist <doc-sci@uiowa.edu>'
```

*Example DSDF: voyager/1/pws/SpecAnalyzer-4s-Efield.dsdf*

The following Juno Ephemeris data source definition file provides the same kinds of information as the DSDF above, but add extra details needed for handling data that is produced at regular intervals. In this case the data are produced by consulting SPICE kernels over the time frame of interest for the Juno Mission. Bold items are required. Since this data stream should not be reduced in the time domain the special flag 'not_reducible' is set. Also this data source requires an extra parameter. In addition to the start point and end point, the program expects as it's third argument the interval between output points. The example below also carries keywords not defined in table 5 above, which is valid. Das2 servers blindly pass unknown keywords out to client programs.

```
description     = 'Juno Solar orbit parameters'
server          = 'http://planet.physics.uiowa.edu/das/das2Server'
reader          = '/opt/project/juno/bin/centos5.x86_64/jephemrdr2 2'
spacecraft      = 'Juno'
spacecraft_id   = 'JO'
das2Stream      = 1
reducer         = 'not_reducible'
requiresInterval = 1
techContact     = 'John Doe <john-doe@uiowa.edu>'
readerSource    = 'https://saturn.physics.uiowa.edu/svn/juno/trunk/ephemeris'
version         = 502
change_01       = '2014-08-11: Switched to Heliographic Inertial coord. frame'

; Use post earth fly-by data for a test range, with a 3-hr "tick" interval
testRange       = '2013-11-01 to 2013-11-02'
testInterval    = 10800
```

*Example DSDF: juno/ephemeris/jephemSun.dsdf*

The following following Juno Magnetometer data source definition file specifies that pre-reduced data at various should be cached.

```
description     = 'Magnetic Field Components in Spacecraft Solar Ecliptic Frame'
techContact     = 'Jane Doe <jane-doe@gsfc.nasa.gov>'
sciContact      = 'Co Invistigator <co-investigator@gsfc.nasa.gov>'

reader          = 'fgm_pds_miscrdr --das2times=scet'
reducer         = 'das2_bin_avgsec'

exampleRange    = '2014-04-08T00:00 to 2014-04-08T01:00'
cacheLevel_00   = 'intrinsic | daily'
cacheLevel_01   = '1 s | daily'
cacheLevel_02   = '60 s | daily'
```

*Example DSDF: juno/fgm/MagComponentsSCSE.dsdf*

## 3.4 Normalizing Reader Parameters

TODO: Write this

# 4 Das 2.2 Stream Format

Typical space physics data sets have a much greater extent in the time dimension than in the various measurement dimensions. For example, the electric field spectrum analyzer on the Voyager missions only have 16 frequency channels, but a spectra have been collected about 4 times a minute for over 35 years! Obviously this data set has a much larger extent in the time dimension than the frequency dimension. Most space physics particles and fields data sets have this property, so das2 streams were designed for delivering time-dependent values in such a way that data reduction in the time domain is easy to accomplish. To do this increasing file offsets correspond with increasing independent data values as illustrated in figure 1 to the right.

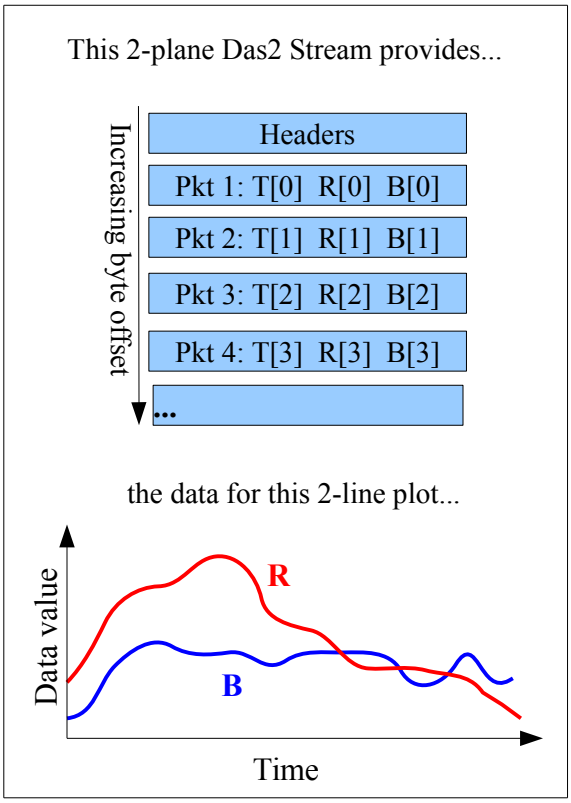



*Figure 1: Stream versus Plot comparison*

A Das 2 stream consist of two types of packets:

- *Header packets*: These packets contain data structure definitions and processing information messages.
- *Data packets*: These contain the actual data, typically in binary form.

Das2 Streams contain 2 to N co-varying vectors. Each data packet provides an X-axis value, with is usually time, and at least one measurement value for each of the independent data values. Though this would seem to restrict the streams to two dimensional data, the X tagged Y scan stream type described in section 4.3 allows this format to provide three dimensional information as well.

Each set of dependent values is called a data plane. For X versus Y streams, a plane can be though of as a single column of data. The stream depicted in Figure 1 above contains two Y planes. The first is the R vector and the second is the B vector.

## 4.1 Header Packet Wrappers

All header packets have a 10 byte prefix consisting of ASCII text characters. Encoded in the prefix is the *Packet ID*, and the Packet Length as defined in the following table.

| **Byte Index:** | **0** | **1-2** | **3** | **4-9** | **10 to Packet Length + 10** |
|---|---|---|---|---|---|
| **Value:** | 0x5B ASCII **[** | Packet ID, zero padded ASCII integer | 0x5D, ASCII **]** | Packet Length*, zero padded ASCII integer | UTF-8 Header Message defining data packets with the same ID as the header |

*This value is *only* equal to the number of characters in the message body *if* the message can be represented in 7-bit safe ASCII text. Typically this is the case, but not always.

Different packet ID's formats are required for different header packets types.

- ASCII **00** – Indicates a Stream Header Packet. A stream always starts with stream header packet. The two bytes have the raw values 0x30, 0x30.
- ASCII **xx** – Indicates a comment packet. Comment packets may be used to provide status information to the end user. Many client programs, such as Autoplot display these text messages to the end user as data continues to load. The two bytes have the raw values 0x78, 0x78.

- ASCII **01** to **99** – Indicates data information, either a data packet header or a data packet itself. The packet ID values for a header packet and it's associated data packet must match. The raw value range for these two bytes is 0x30 to 0x39, though the combination 0x30,0x30 is not allowed.

## 4.2 Stream Header Packet

Every das2 stream must start with a stream header packet. The stream header is an XML message with the top level element `<stream>`. The character encoding for Stream Header Packets is UTF-8. A typical stream header packet follows.

```
[00]000108<stream version="2.2">
  <properties Datum:xTagWidth="128.000000 s" double:zFill="-1.0e+31" />
</stream>
```

The following table defines the elements that make up a header packet.

| Element | | Notes |
|---|---|---|
| **<stream>** | | Must always be contained in a header packet with packet id "00" |
| *attribute* | **compression** | If set to the value "`deflate`" then then rest of the stream after the stream header packet has been compressed using the the deflate method from the ZLIB compression library |
| *attribute* | **version** | For streams conforming to this document the version attribute value should always be `"2.2"`. |
| <stream> **<properties>** | | Defines useful properties of this stream. |
| *attribute* | **String:renderer** | Used to suggest a plot type to the client program, values may be '`spectrogram`', '`symbolLine`', '`stackedHistogram`', & '`waveform`'. |
| *attribute* | **String:title** | If present this provides a title for plots generated from this dataset. |
| *attribute* | **String:summary** | Used to describe an entire Das2 stream. |
| | **DatumRange: xCacheRange** | This important attribute provides clients with the coverage range of the data returned. Clients use this information, along with the `xCacheResolution`, to decide if a Das2 server must be contacted for new data after a zoom operation, or if rescaling data already provided is is sufficient. Proper use of CacheRange and CacheRelosultion tags can significantly increase the performance of Das2 servers and clients. |
| *attribute* | **Datum: xCacheResolution** | Used with `xCacheRange`, this attribute provides the resolution of a stream in the X dimension. This attribute should always be set for data sources with a known resolution, such as ephemeris readers, or data passed through bin-averagers. If this attribute not present, the data are considered to be at intrinsic resolution. Thus many readers will not need to set this attribute. |
| *attribute* | **String:xFormat** | If preset provides a |
| *attribute* | **String:xLabel** | If preset provides an x-axis label for plots generated from this dataset. Labels may contain formatting flags which allow for subscripts and superscipts, see section 4.2.1 below for details. |
| *attribute* | **boolean: xMonotonic** | If present allows the client to begin processing data before receiving the entire stream. |

| Element | Notes | |
|---|---|---|
| *attribute* | **DatumRange:xRange** | Useful for setting X scale in client programs. The format of range items is "*START* `to` *END UNITS*", where *START* and *END* are the inclusive beginning and **exclusive** ending values, and *UNITS* are the units string, if applicable. |
| *attribute* | **String:xSummary** | A brief description of the X direction values |
| *attribute* | **Datum:xTagWidth** | Defines the minimum gap in the X direction across which client programs should not interpolate. |
| *attribute* | **double:xValidMax** | If present provides a maximum possible valid data value for X axis values. Data above this value are not displayed. |
| *attribute* | **double:xValidMin** | If present provides a minimum possible valid data value for X axis values. Data below this value are not displayed. |
| *attribute* | *DatumRange: yCacheRange* | Reserved for future use with data sources that support "area" queries, i.e. min X,Y to max X,Y. |
| *attribute* | *Datum: yCacheResolution* | Reserved for future use with data sources that support "area" queries, i.e. min X,Y to max X,Y. |
| *attribute* | **double:yFill** | Specifies a fill value for missing Y data values. Defaults to -1e31 if not specified. |
| *attribute* | **String:yLabel** | If preset provides an y-axis label for plots generated from this dataset. |
| *attribute* | **DatumRange:yRange** | Useful for setting Y scale in client programs. See the `DatumRange:xRange` attribute for formatting details |
| *attribute* | **String:yScaleType** | May be one of the strings "log" or "linear" |
| *attribute* | **String:ySummary** | A brief descripton of the Y direction values |
| *attribute* | **Datum:yTagWidth** | Defines the minimum gap in the Y direction across which client programs should not interpolate. |
| *attribute* | **double:yValidMin** | If present provides a minimum possible valid data value for Y axis values. Data below this value are not displayed. |
| *attribute* | **double:yValidMax** | If present provides a maximum possible valid data value for Y axis values. Data above this value are not displayed. |
| *attribute* | **double:zFill** | If present, defines the data Z values which should be read as a fill values. By default the constant -1.0e31 is used as the fill value. |
| *attribute* | **String:zLabel** | If present provides a z-axis label for plots generate from this dataset. |
| *attribute* | **DatumRange:zRange** | Useful for setting Z scale in client programs. See the `DatumRange:xRange` attribute for formatting details |
| *attribute* | **String:zScaleType** | May be one of the strings "log" or "linear" |
| *attribute* | **String:zSummary** | A brief descripton of the Z direction values |
| *attribute* | **double:zValidMin** | If present provides a minimum possible valid data value for Z axis values. Data below this value are not displayed. |
| *attribute* | **double:zValidMax** | If present provides a maximum possible valid data value for Z axis values. Data above this value are not displayed. |

| Element | Notes | |
|---|---|---|
| *attribute* | **boolean:*AnyID***<br>**String:*AnyID***<br>**double:*AnyID***<br>**Datum:*AnyID***<br>**DatumRange:*AnyID***<br>***int:AnyID***<br>***Time:AnyID***<br>***TimeRange:AnyID*** | In addition to the named items below, generic metadata may be provide, though client programs will not necessarily know what to do with these extra properties, other than to store them for presentation to the end user.<br><br>Examples of using this feature:<br>`String:maintainer="some-person@uiowa.edu"`<br>`Time:creation_time="2014-04-23T00:30:23"` |

*Table 6: Das2.2 Stream Header XML Elements*

### 4.2.1 Label Values

The xLabel, yLabel and zLabel properties define text labels for plot axes. The following formatting strings may be embedded within the labels to alter the placement and appearance of the text.

```
!A  shift up one half line
!B  shift down one half line  (e.g.  !A3!n-!B4!n is 3/4).
!C  newline
!D  subscript 0.62 of old font size.
!U  superscript of 0.62 of old font size.
!E  superscript 0.44 of old font size.
!I  subscript 0.44 of old font size.
!N  return to the original font size.
!R  restore position to last saved position
!S  save the current position.
!K  reduce the font size. (Not in IDL's set.)
!!  the exclamation point (!)
```

Case is not important, so "`!A`" and "`!a`" have the same effect.

### 4.2.2 Format Strings

Many Das2 Stream properties provided hints on how data should be formatted if presented in a graphical form, only three properties, xFormat, yFormat and zFormat properties provide hints on how data should be printed as ASCII strings.

* The following C-style conversion specifiers work for all types:

%f

%e

* Time types typically are presented as multi-part fields. The following conversion specifiers are understood for data indicated as being a time type via a units property of `us2000,t2000,us1980, t1970, mj1958, mjd, cdfEpoch,` tt2000.

The following mostly C-style conversion specifiers work for time values:

%Y - A 4-digit year

%m - A 2-digit month

%d - A

Characters that are not part of an understood format specifier are to be repeated in the output.

## 4.3 Data Header Packets

Following the stream header packet is at least one data header packet. Data packets of a particular type must be proceeded in the stream by a header packet defining it's contents. There is no requirement that all header packets types be placed at the start of the stream. It is merely sufficient that each type of data packet is defined somewhere before it is encountered. The character encoding for Data Header Packets is UTF-8.

There are two basic data packet types:

- X tagged: For this packet type the co-varying Y vectors are merely correlated in the X-units and each may have different units of measure. Data of this type are defined by including 1-N **<y>** child elements under **<packet>** element.
- X tagged Y Scan: For this packet type the co-varying Y vectors all provide measurements in the same units as well as being correlated in the X-units. Data of this type are defined by including a <yscan> child element under the <packet> element.

### *4.3.1 X Tagged Stream Example*

The das2 stream fragment below is taken from the Van Allen Probes Ephemeris data source. This is an **X tagged** stream providing four vectors of spacecraft A position information with respect to time. In each data packet, the first value is spacecraft event time, the second is the corresponding radial position in $R_E$, the third is the Magnetic Latitude, the forth is the Magnetic Local Time and the fifth is the L-shell value. Each “vertical” data vector provides “Y” values in different units. Essentially this stream contains four correlated 2-D datasets in parallel.

```
[00]000102<stream version="2.2">
<properties DatumRange:xRange="2013-001T01:00:00 to 2013-01:10:00"/>
</stream>
[01]000424<packet>
  <x type="time23" units="us2000"></x>
  <y type="ascii11" name="radius" units="">
    <properties String:yLabel="R!DE!N" />
  </y>
  <y type="ascii11" name="mag_lat" units="">
    <properties String:yLabel="MLat" />
  </y>
  <y type="ascii11" name="mag_lt" units="">
    <properties String:yLabel="MLT" />
  </y>
  <y type="ascii11" name="l_shell" units="">
    <properties String:yLabel="L" />
  </y>
</packet>
:01:2013-001T01:00:00.000  5.782e+00 -1.276e+01  3.220e+00  6.079e+00
:01:2013-001T01:01:00.000  5.782e+00 -1.274e+01  3.232e+00  6.077e+00
:01:2013-001T01:02:00.000  5.781e+00 -1.272e+01  3.244e+00  6.076e+00
```

### *4.3.2 X Tagged Y Scan Stream Example*

The following example contains a single 3-D dataset. It's a cut down version of the Voyager PWS Spectrum analyzer dataset. Here the X dimension is again time, but the data are Z-values. The Y dimension values are provided *in the header* instead of in the data packets. There is one Y value in the `<yscan>` element for each column of data values, this is much more efficient then repeating the Y values in each data packet.

```
[00]000248<stream version="2.2">
  <properties xMonotonic="true" xLabel="Time (s)" yLabel="Frequency (s!U-1!N)"
              zLabel="Electric Field (V m!U-1!N)" title="Voyager 1 PWS SA"
              Datum:xTagWidth="16.0 s" double:zFill="0.0" />
</stream>
[01]000201<packet>
  <x type="time24" units="us2000" ></x>
  <yscan nitems="5" type="ascii9" yUnits="Hz" name="" zUnits="V/m"
         yTags="10.0,17.8,31.1,56.2,100.0">
    <properties  />
  </yscan>
</packet>
:01:2012-01-01T12:56:22.792 1.91e-06 8.92e-07 7.80e-07 6.04e-07 2.43e-07
:01:2012-01-01T12:56:38.792 1.91e-06 8.00e-07 8.47e-07 5.42e-07 4.36e-07
:01:2012-01-01T12:56:54.792 1.98e-06 4.63e-07 7.64e-07 7.56e-07 5.09e-07
:01:2012-01-01T12:57:10.792 1.91e-06 5.46e-07 7.97e-07 5.42e-07 6.55e-07
:01:2012-01-01T12:57:26.792 1.86e-06 6.80e-07 8.53e-07 4.40e-07 3.64e-07
```

X tagged Y scan streams work well for cubic dataset, but cannot handle X,Y,Z scatter data without a ridiculous number of fill values, to plot scatter data use streams with <z> planes.

### *4.3.3 X Y Z Scatter Data Example*

As a catchall to handle data that are not nicely aligned on a 2-D grid <x><y><z> scatter data streams are supported. This example provides plasma density measurements over Mars from the MARSIS experiment.

```
[00]000497<stream version="2.2">
  <properties String:title="MARSIS Plasma and Magnetic Field Parameters"
              xLabel="LONG" xSummary="West Longitude"
              yLabel="LAT" double:zFill="-1.0"
              ySummary="This value is Planetocentric Latitude, but for radar
                        sounding Planetographic Latitude is more accurate.
                        This value is used to allow for direct comparison with
                        other Mars Express data sets." />
</stream>
[01]000587<packet>
  <x type="ascii8" units="degrees"></x>
  <y type="ascii8" units="degrees"></y>
  <z type="ascii10" name="" units="kHz">
    <properties zLabel="F!Dpe!N" zSummary="Electron Plasma Frequency" />
  </z>
  <z type="ascii10" name="dens" units="cm**-3">
    <properties zLabel="N!De!N" zSummary="Electron Plasma Density"/>
  </z>
  <z type="ascii10" name="fce" units="kHz">
    <properties zLabel="F!Dce!N" zSummary="Electron Cyclotron Frequency" />
  </z>
  <z type="ascii10" name="mag" units="nT">
    <properties zLabel="B!Dmag!N" zSummary="B-Field Magnitude" />
  </z>
</packet>
:01: 186.49  -36.82     -1.0      -1.0  4.67e-01  1.67e+01
:01: 186.49  -36.49     -1.0      -1.0  8.77e-01  3.13e+01
:01: 186.49  -36.16     -1.0      -1.0  9.11e-01  3.25e+01
:01: 186.49  -35.83     -1.0      -1.0  1.06e+00  3.80e+01
:01: 186.50  -35.50     -1.0      -1.0  9.94e-01  3.55e+01
```

## 4.4 Data Header Reference Table

Table 7 below defines the XML elements used in Das 2.2 header packets.

| Element | | Notes |
|---|---|---|
| **<packet>** | | This element's children define one data packet type. For each data packet type on of these elements must be defined. Each packet header contains one, and only one packet element. The top level element is merely a grouping element and contains no attributes. The <x> child element is required and one or more <y> or <yscan> elements. |
| <packet><br>**<x>** | | Defines the x-axis values for a data packet. These values are assumed to be the first entry in each data packet. The x-axis dimension is also the streaming dimension. Any data reduction handled by the das2server is performed in this dimension. The x-axis are almost always in units of time. |
| *attribute* | **base** | If defined this provides the epoch time as an ASCII ISO-8601 time string. Use one of "hr", "min", "s" "microseconds","nanoseconds", or "days" for the units attribute value if a base is specified. |
| *attribute* | **type**<br>(required) | The data format for the x-axis data elements, must be one of sun_real8, sun_real4, little_endian_real8, little_endian_real4, timeN, and asciiN, where N is the number of characters in an ASCII time string, for example time23, or ASCII data value, for example ascii10. |
| *attribute* | **units**<br>(required) | The "units" of the X axis values. These are not just units as typically used in physics but are instead units plus an implied 0 point. See section 4.4.1 for known values of the units attribute. Note: Even if the type attribute indicates that the X values are ASCII ISO-8601 time strings this attribute must be set for internal conversions within reader programs. |
| <packet><br><x><br>**<properties>** | | Das2 properties cascade. Thus to override a general packet property, for this <x> plane, put property values in this element. See the <properties> element in Table 6. |
| <packet><br>**<y>** | | Defines a y-axis value for a data packet. Note that a packet may have any number of <y> planes, as long as it does **not** have a <z> plane. Typically <y> planes are used for the values in a line plot, or as the second dependent variable for <x>, <y>, <z> scatter data. |
| *attribute* | **type**<br>(required) | See the description for the type attribute in the **<x>** element. |
| *attribute* | **name**<br>(usually-required) | Provides a name for the data plane, follows the formation rules for C identifiers. Unless a stream contains <x><y><z> scatter data the name field is required. For 3-axis scatter data streams Y is an independent data axis, and the name attributed is not required. |
| *attribute* | **units**<br>(required) | The units of the this Y-vector's data values. See section 4.4.1 for known values of the units attribute. |
| <packet><br><y><br>**<properties>** | | Das2 properties cascade. Thus to override a general packet property, for this <y> plane, put property values in this element. See the <properties> element in Table 6: Das2.2 Stream Header XML Elements. There are a couple predefined properties that only make sense at the level of a data plane. These are defined below. |

| Element | Notes | |
|---|---|---|
| | **String:operation** | If the data in this <y> plane were derived via some mathematical operation on data from another plane, provide the name of the operation here: Currently understood values are:<br>`BIN_AVG, BIN_MAX, BIN_MIN, DFT_POWER` |
| | **String:source** | If the data in this <y> plane were derived via some mathematical operation on data from another plane, provide the name of the other plane here, this allows client programs to plot derived data values together on the same plot for the same upstream source. |
| <packet><br>**<yscan>** | Defines a set of co-varying vectors with the same magnitude units. For this data type the vector Y index is associated with some physical value, typically frequency. | |
| *attribute* | **name**<br>(required) | Provides a name for the data plane. Follows the rules for C-identifers. |
| *attribute* | **nitems**<br>(required) | The number of co-varying vectors. Note that each data packet represents one slice across a constant number of vectors. |
| *attribute* | **type**<br>(required) | See the description for the `type` attribute in the **<x>** element. |
| *attribute* | **yTags**<br>(optional) | Use this attribute to associate the vector number with a physical value. The value portion is a comma separated list of ASCII text values, for example:<br>`yTags="12.5, 15.5, 16.7"`<br><br>If neither the `yTags` or `yTagInterval` attributes are provided, then the y values are assumed to be the same as the data index, i.e. `0` to `(nitems - 1)` |
| *attribute* | **yTagInterval**<br>(optional) | For Y values that are a series of points with a fixed value separating each point individual yTags need not be specified. Instead, the interval between points may be provided using this attribute. Waveforms and operations on waveforms typically have linearly spaced y values. |
| *attribute* | **yTagMin**<br>(optional) | Used to provide the y value for the 0th index of the yscan. This attribute is optional with the following defaults:<br><br>- If `yTagInterval` is absent, this attribute is ignored<br><br>- if `yTagInterval` is given but not `yTagMax`, defaults to: 0<br><br>- if `yTagInterval` and `yTagMax` are given, defaults to:<br>`yTagMax - (nitems - 1) * yTagInterval`. |
| *attrubute* | **yTagMax**<br>(optional) | Used to provide the y value for last index of the yscan. This attribute is optional with the following defaults:<br><br>- If `yTagInterval` is absent, this attribute is ignored<br><br>- if `yTagInterval` is given but not yTagMin, defaults to:<br>`(nitems - 1)*yTagInterval`<br><br>- if `yTagInterval` and `yTagMin` are given, defaults to:<br>`yTagMin + (nitems - 1) * yTagInterval` |

| Element | Notes | |
|---|---|---|
| *attribute* | **yUnits** | Provides the physical units for the yTags values. The most common values are: "Hz", "KHz" and "MHz", see section 4.4.1 for known values of the units attribute. |
| *attribute* | **zUnits** | Provides the physical units for the vector data values, for example: "nT/Hz". See section 4.4.1 for known values of the units attribute. |
| <packet><br><yscan><br>**<properties>** | Das2 properties cascade. Thus to override a general packet property, for this <yscan> plane, put property values in this element. See the <properties> element in Table 6: Das2.2 Stream Header XML Elements. There are a two pre-defined properties that only make sense at the level of a <yscan> data plane. These are defined below. | |
| *attribute* | **boolean:peaks** | If true then clients are to treat this extra <yscan> as the peaks for the first, unnamed, <yscan>. |
| *attribute* | **boolean:weights** | If true then clients are to treat this extra <yscan> as the weights for the first unnamed <yscan>. |
| <packet><br>**<z>** | Defines a set of z values. Must be accompanied by a **single** <x> plane and a single <y> plane. Any number of <z> planes are allowed. | |
| *attribute* | **name**<br>(required) | Provides a name for the data plane, this should follow the rules for C identifiers with the relaxation that he first character can be a digit. |
| *attribute* | **type**<br>(required) | See the description for the type attribute in the **<x>** element. |
| *attribute* | **units**<br>(required) | The units of the this Z-vector's data values. See section 4.4.1 for known values of the units attribute. |
| <packet><br><z><br>**<properties>** | Das2 properties cascade. Thus to override a general packet property, for this <z> plane, put property values in this element. See the <properties> element in Table 6: Das2.2 Stream Header XML Elements. | |

*Table 7: Das2.2 Packet Header XML Elements*

### *4.4.1 Unit Values*

Known values for this attribute are which include an epoch time are:

- us2000 – Microseconds since midnight January 1st 2000, ignoring leap seconds
- t2000 – Like us2000 but in units of seconds, ignoring leap seconds, i.e. every day is the same length
- us1980 – Microseconds since midnight January 1st 1980, ignoring leap seconds
- t1970 – Seconds since midnight January 1st 1970, ignoring leap seconds.
- mj1958 – Days since midnight 1958-01-01, more accurately Julian day – 2436204.5.
- mjd – Days since midnight November 17, 1858.
- cdfEpoch – milliseconds since 01-Jan-0000 TODO: What the heck is year 0?
- tt2000 – nanoseconds since 01-Jan-2000 including leap seconds, may be transmitted as an 8-byte integer.

Other values for units are:

"dB", "radian", "degrees", "celcius degrees", "fahrenheit degrees", "centigrade", "deg

F", "hr", "min", "s", "ms", "microseconds", "nanoseconds", "days", "bytes/s", "KBytes/s", "bytes", "KBytes", "Hz", "kHz", "MHz", "eV", "cm!a-3!n, K, cm/s", "V!a2!nm!a-2!nHz!a-1", "V/m", "W/m!a-2!n", "inch", "m", "km".

Cleaning up the unit handling is a priority for the Das2 developers.

## 4.5 Comment Packets

Comment packets use the same wrapper format as Header Packets with the exception of the Packet ID field. Instead of an integer ID consists of the bytes 0x78, 0x78, i.e. ASCII '**xx**'. Here's an example stream with embedded comments.

```
[00]000067<stream version="2.2"><properties monotonicXTags="true"/></stream>
[xx]000053<comment type="taskSize" value="100" source=""/>
[xx]000138<comment type="log:info" value="Input:T120101.DAT" source="vgpwrdr"/>
[01]000550<packet>
  <x type="time24" units="us2000" ></x>
  <yscan nitems="5" type="ascii10" yUnits="Hz" name="" zUnits="V/m"
         yTags="10.0,17.8,31.1,56.2,100.0">
  </yscan>
</packet>
:01:2012-01-01T12:56:06.792 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00
[xx]000053<comment type="taskProgress" value="1" source=""/>
:01:2012-01-01T12:56:22.792 1.91e-06 8.92e-07 7.80e-07 6.04e-07 2.43e-07
:01:2012-01-01T12:56:54.792 1.98e-06 4.63e-07 7.64e-07 7.56e-07 5.09e-07
[xx]000053<comment type="taskProgress" value="2" source=""/>
:01:2012-01-01T12:57:26.792 1.86e-06 6.80e-07 8.53e-07 4.40e-07 3.64e-07
:01:2012-01-01T12:57:58.792 1.98e-06 4.63e-07 8.36e-07 8.57e-07 3.23e-07
[xx]000138<comment type="log:info" value="Input:T120102.DAT" source="vgpwrdr"/>
:01:2012-01-01T12:58:14.792 1.98e-06 6.05e-07 7.80e-07 6.65e-07 3.23e-07
```

Comment packets contain a single `<comment>` element who's attributes are defined in the table below. The character encoding for Comment Packets is UTF-8.

| Element | | Notes |
|---|---|---|
| **<comment>** | | Allows a das2 stream source to add processing information and other information into the stream. Probably the most valuable use to provide progress updates that can be used for progress bars by the client software. |
| *attribute* | **source** (required) | A text string indicating the source of the message. This is useful for diagnostic messages as the stream may be processed by more than one program on it's way to the client. |
| *attribute* | **type** (required) | Specifies the type of the comment. At present two attribute values are understood:<br>● `"log:info"` – Indicates that the `value` attribute will provide a human readable string for display by the client program during the data load.<br>● `"taskSize"` – Indicates that the value attribute will provide an integer greater than 0 the overall size of a task as an integer. Place one of these comments into the stream first and then follow by `type="taskProgress"` comments to update overall completion level.<br>● `"taskprogress"` – Indicates that the value attribute will provide an integer from 0 to taskSize. Use this comment to set the processing completion level. |
| *attribute* | **value** (required) | The value of the `value` attribute varies based on the `type` attribute value. |

## 4.6 Exception Packets

Exception packets use the same wrapper format as Header Packets with the exception of the Packet ID field. Instead of an integer ID consists of the bytes 0x78, 0x78, i.e. ASCII '**xx**'. Here's an example stream that terminates in an exception. In this case a file IO error.

```
[00]000485<stream version="2.2"><properties monotonicXTags="true"/></stream>
[01]000544<packet>
  <x type="time24" units="us2000" ></x>
  <yscan nitems="16" type="ascii10" yUnits="Hz" name="" zUnits="V/m"
         yTags="10.0,17.8,31.1,56.2,100.0">
  </yscan>
</packet>
:01:2012-01-01T12:56:06.792 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00
:01:2012-01-01T12:57:26.792 1.86e-06 6.80e-07 8.53e-07 4.40e-07 3.64e-07
:01:2012-01-01T12:57:58.792 1.98e-06 4.63e-07 8.36e-07 8.57e-07 3.23e-07
:01:2012-01-01T12:58:14.792 1.98e-06 6.05e-07 7.80e-07 6.65e-07 3.23e-07
[xx]000102<exception type="IOError"
  message="Directory /opt/project/voyager/pds/VGPW_0101 does not exist" />
```

Comment packets contain a single `<comment>` element who's attributes are defined in the table below. The character encoding for Execption Packets is UTF-8.

| Element | | Notes |
|---|---|---|
| **`<exception>`** | | Allows a das2 stream source to end with an error message that is displayable by an end user client. |
| *attribute* | **`type`** (required) | The type of error. The following error types are understood:<br>● `"NoDataInInterval"`– The data selection parameters excluded all available data<br>● `"IllegalArgument"` – One or more of the parameters given to the reader is not valid. For example, if an Antenna selection parameter could take either `"Bx"` or `"Ex"` and the program received `"doggy"` then this would be an appropriate exception to return.<br>● `"ServerError"` - There is an internal problem in the Das2 server or one of it's readers. This could be cause by a bad disk, file-system permission problems, bad DSID files, etc. There is nothing the user can do from the client side to fix the problem.<br>● ANYSTRING – Any other error type string may but this is merely for documentation or for use by custom clients. |
| *attribute* | **`message`** (required) | The message string for the error. |

## 4.7 Data Packets

Data packets are a mixture of values whose interpretation is provided by the proceeding header packet **with the same packet id**. Das2 Stream data packets are fixed length. All packets with the same ID must have the same length, unless the packet is redefined in the middle of the stream. If new packet header is encountered on a stream it's definition replaces the previous definition with the same packet ID. Upon re-definition, all packets must now conform to the new length requirement. The following table summarizes the data packet format.

| Byte Index | 0 | 1-2 | 3 | 4-9 | 10 to Packet Length + 10 |
|---|---|---|---|---|---|
| **Content** | 0x3A ASCII ':' | Packet ID. Zero padded ASCII integer | 0x3A ASCII ':' | Packet Length. Zero padded ASCII integer | **1-N Data values.** Data value format is defined by the header with the same packet ID |

## 4.8 Macro Substitution in Header Packets

Das2 Stream data may pass through multiple programs from the time they are emitted from a reader until being read by a client program. Some of the keywords in the stream, such as the xCacheResolution are updated as needed along the way. To make streams information more transparent to end users, macro expansions may be added to string attributes in the stream headers. Any parameter non-self-referencing value may be substituted. The following example header will help to illustrate the concept.

```
[00]000494<stream compression="none" version="2.2" >
  <properties String:title="B-Field Magnitude!c(%{xCacheResolution} averages)"
              String:xLabel="SCET (UTC)"
              Datum:xTagWidth="60.000000 s"
              String:yScaleType="linear"
              DatumRange:xCacheRange="2012-03-09 to 2012-03-10 UTC"
              Datum:xCacheResolution="60.000000 s"
              String:sourceId="das2_bin_avgsec"/>
</stream>
[01]000843<packet>
  <x type="time25" units="us2000">
  </x>
  <y name="mag" type="little_endian_real4" units="nT">
    <properties String:ySummary="The magnitude of the magnetic field."
                String:yLabel="Magnitude (nT)"/>
  </y>
</packet>
```

Notice the bold text `%{xCacheResolution}` in the title attribute. This macro will expand to whatever value the attribute Datum:xCacheResolution happens to have when the client program receives the data. The format rule for Das2 Stream macros are:

**"%{" + ANY_PROPERTY_NAME + "}"**

There are a few items to remember when using macro substitutions in Das2 stream and packet headers:

1. The property type is not required, just it's name.
2. Substitutions work for any stream attribute, not just the standard items defined in section 4.2.
3. Since Das2 stream attributes cascade, the nearest attribute value is substituted.
4. Self referential macros are not allowed, i.e. don't use `%{title}` in a `title` attribute value.

The example above is taken from the output of the standard Das2 PyServer program das2_bin_avgsec, which automatically adds the %{xCacheResolution} macro to the stream header as the data pass through.

# 5 QStream Format

QStreams were introduced after QDataSets were introduced with the Das2 application Autoplot. QDataSet is the more flexible data model that allows data in more forms to be represented, and roughly mirrors the structure of data in a CDF file. QStreams also represent this data, but like Das2Streams allow the data to be streamed so that processing can be done on the stream as it is received. As with Das2Streams, descriptive XML headers are mixed with binary or ASCII data packets.

#### Differences between QStreams and Das2 Streams

A few things change with QStreams, namely that any number of datasets can be sent on a stream, and the stream header identifies which of the datasets is the default dataset to be interpreted. Instead of a dataset having multiple planes, there are simply multiple datasets and the semantics of QDataSet connect them. Also new, the entire stream must use the same byte order (endianness), and streams can no longer be compressed. Data can be encoded within the header elements, or encoded along with the other data packets. (Das2Streams had the problem that packets would be large when there were many ytags.)

## 5.1 Stream Descriptor

A valid QStream starts with a header. This identifies the endianness of the stream and the default dataset. The default byte order is big endian, and the packet will contain the tag “byte_order” to set the order.

```
[00]000096<?xml version="1.0" encoding="UTF-8"?>
<stream byte_order="little_endian" dataset_id="MinMax"/>
```

This indicates the byte order of the stream will be little endian, and the default dataset will have the id “MinMax.” Optionally but encouraged, the xml encoding should be specified as well. Note too that all QStreams start with [00], as do all Das2Streams. The six integers following the [00] tag indicate the descriptor xml is 96 bytes long. A stream can have any number of stream descriptors, but each stream descriptor must have the same default dataset_id and byte_order. Last, though this is never tested, it should be said that the byte sequence “[00]dddddd<dddddd bytes>[” must never appear within the data or packet contents.

<TODO: reference to a schema for the stream descriptor> <TODO: high-rank datasets>

## 5.2 Packet Descriptors

A packet descriptor contains XML with the tag packet containing any number of qdataset packets. Each qdataset tag has an id and a rank attribute. Each contains a properties tag the identify the properties of the dataset. Last, each contains a values tag that defines either how the data will be encoded in each packet, or contain the values in-line.

In the following packet descriptor, two qdatasets are described, MinMax and ds_0. ds_0 contains just one double per packet (packet is a record encoded on the stream) These records are combined as they come in to form a rank 1 dataset of length N where N is the number of [01] packets. MinMax contains two doubles, which are combined to make a rank 2 dataset that is [N,2].

```
[01]001079<?xml version="1.0" encoding="UTF-8"?>
<packet>
  <qdataset id="ds_0" rank="1">
    <properties>
      <property name="UNITS" type="units" value="cdfTT2000"/>
      <property name="FILL_VALUE" type="Double" value="-1.0E38"/>
      <property name="LABEL" type="String" value="Epoch+ns"/>
    </properties>
    <values encoding="double" length=""/>
```

```
  </qdataset>
  <qdataset id="MinMax" rank="2">
    <properties>
      <property name="BINS_0" type="String" value="min,maxInclusive"/>
      <property name="DEPEND_0" type="qdataset" value="ds_0"/>
      <property name="UNITS" type="units" value="counts"/>
      <property name="SCALE_TYPE" type="String" value="linear"/>
      <property name="NAME" type="String" value="MinMax"/>
      <property name="USER_PROPERTIES">
        <map>
          <entry key="count" type="Integer" value="65536"/>
        </map>
      </property>
    </properties>
    <values encoding="double" length="2"/>
  </qdataset>
</packet>
```

Encodings can be float, double, int, short, byte, long, ascii10, time17, etc. With ascii10, the 10 characters are parsed as a double. With time17, the characters are parsed as an ISO8601 string.

```
[00]000067<?xml version="1.0" encoding="UTF-8"?>
<stream dataset_id="ds_0"/>
[02]000485<?xml version="1.0" encoding="UTF-8"?>
<packet>
  <qdataset id="ds_2" rank="1">
    <properties>
      <property name="FILL_VALUE" type="Double" value="-1.0E38"/>
      <property name="CADENCE" type="rank0dataset" value="1.5351
                units:UNITS=logERatio String:SCALE_TYPE=log"/>
      <property name="SCALE_TYPE" type="String" value="log"/>
    </properties>
    <values values="10.0,46.41588833612777,215.44346900318823,1000.0"/>
  </qdataset>
 </packet>
[01]000844<?xml version="1.0" encoding="UTF-8"?>
<packet>
  <qdataset id="ds_1" rank="1">
    <properties>
      <property name="UNITS" type="units" value="us2000"/>
      <property name="CADENCE" type="rank0dataset" value="6.0E7
                units:UNITS=microseconds"/>
      <property name="MONOTONIC" type="Boolean" value="true"/>
    </properties>
    <values encoding="time17" length=""/>
  </qdataset>
  <qdataset id="ds_0" rank="2">
    <properties>
      <property name="DEPEND_0" type="qdataset" value="ds_1"/>
      <property name="DEPEND_1" type="qdataset" value="ds_2"/>
      <property name="FILL_VALUE" type="Double" value="-1.0E31"/>
      <property name="QUBE" type="Boolean" value="true"/>
    </properties>
    <values encoding="ascii10" length="4"/>
  </qdataset>
</packet>
:01:2013-04-04T00:00   -0.0325    0.0185    0.0014   -0.0008
:01:2013-04-04T00:01   -0.0314     0.019    0.0167   -0.0002
:01:2013-04-04T00:02   -0.0048   -0.0486   -0.2026   -0.0489
```

```
:01:2013-04-04T00:03   -0.0051   -0.0489   -0.2026   -0.0489
:01:2013-04-04T00:04    0.0014   -0.0002    0.0166   -0.0002
```

## 5.3 Exceptions

Like the Das2Streams, exceptions can be represented on the stream as well. This enables the server to always return a stream, even when there is an error condition (such as no data).

```
[xx]000155<?xml version="1.0" encoding="UTF-8"?>
<exception type="NoDataInInterval" message="No data found in interval.
           Last data found at 2012-02-01"/>
```

## 5.4 Examples

More examples of streams can be found at:

http://www.jfaden.net:8080/hudson/job/autoplot-test013/lastSuccessfulBuild/artifact/

# Appendix A: Das 2.3 Client - Server Interface

*Das 2.2 is the current stable version. The basic interface and tools have been in use for over 10 years and work well for time series data. Das 2.3 is an expansion of this interface to handle arbitrary data indices and to make more meta-data available to clients.*

The purpose of a Das2 server is to listen for queries and provide responses in a standardized format. Though there are many moving parts behind the scenes, client programs need only implement a Client – Server Interface in order to find and retrieve data.

Two data query protocols are supported by Das 2.2 compliant servers. The legacy 2.1 protocol requires all data to be retrieval by time range. See section 2, for details on the legacy protocol. This section defines the newer 2.2 protocol which allows reader programs to retrieve data via arbitrary parameters. Thus a Das 2.2 client program my query for data by orbit number, or longitude and latitude, or any other parameter supported by the data source.

## A.1: Relative URLs

Das 2.1 and Das 2.2 queries are supported using different relative URLs from the root server path. Table 8 below summaries the different paths used for each request type.

| Request | 2.1 URL and response | 2.2 URL and response |
|---|---|---|
| **None** (i.e get root) | *no relative URL*<br><br>HTML intro page, no links | *no relative URL*<br><br>XML intro with style sheet, contains top level data links and peer links |
| **peers** | `?server=peers`<br><br>XML message with stylesheet containing peer list | `/server/peers`<br><br>XML message with style-sheet containing peer list |
| **list** | `?server=list`<br><br>Plain text listing of data sources, one per line | `/server/list`<br><br>XML message with style-sheet containing full recursive data set list |
| **discover** | `?server=discover`<br><br>Plain text listing of data sources which contain an example range | *not supported* |
| **describe** | `?server=dsdf&dataset=A/B/C.dsdf`<br><br>IDL syntax DSDF file, see section 3.2. | `/data/A/B/C`<br><br>XML DSID file, see section Error: Reference source not found |
| **list level** | *not supported* | `/data/A/B/`<br><br>XML data set list at this level |
| **dataset** | `?server=dataset&dataset=A/B/C.dsdf&`**`DAS2.1_QUERY_STRING`**<br><br>A das2Stream or QStream | `/data/A/B/C?`**`DAS2.2_QUERY_STRING`**<br><br>A das2Stream or QStream |
| **search** | *not supported* | TBD |

*Table 8: Das 2.1 and 2.2 URL comparisons*

The 2.2 URL scheme was devised to make it easier for common web-browsers and search engines to navigate the dataset hierarchy.

## A.2: Describe Request

The query parameters supported by a data-source are defined in the DSID XML message delivered for a **`describe`** operation. This operation is handled as a standard HTTP get on the data source URL, with no query parameters for example:

**Describe Request**

```
GET /data/van-allen-probes/A/emfisis/L2/EmfisisWFRwaveforms HTTP/1.0
Host: das2.physics.uiowa.edu
Accept: *
```

and the DSID file with a style sheet is returned as the response...

**Describe Response**

```
HTTP/1.1 200 OK
Date: Tue, 23 Jul 2013 21:10:53 GMT
Server: Apache/2.0.63
Connection: close
Content-Type: text/xml; charset=UTF-8

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="/server/dsid-stylesheet.xsl"?>
<dasDSID xmlns="http://www.das2.org/dsid/0.2" name="EmfisisWFRwaveforms">

  <summary>Van Allen Probes 35 kHz Waveform data from L2 CDF files.</summary>

  <selectors>
    <param key="scet" name="SCET" min="2010-09-01" format="DATETIME">
      <summary>Spacecraft Event Time</summary>
      <description>
        Any ASCII time string parseable by Larry Granroth's parsetime algorithm,
        including ISO 8601 date times.
      </description>
    </param>

    <enum key="sensor" name="Sensor">
      <summary>Select one of 6 input sensors</summary>
      <description>
      EMFISIS is connected to 3 electric dipole antennas and 3 magnetic search coils.
      </description>
      <items>
        <item value="Bu" summary="Magnetic field U component sampled at 35 kHz" />
        <item value="Bv" summary="Magnetic field V Component sampled at 35 kHz" />
        <item value="Bw" summary="Magnetic field W Component sampled at 35 kHz" />
        <item value="Eu" summary="Electric field U component sampled at 35 kHz" />
        <item value="Ev" summary="Electric field V Component sampled at 35 kHz" />
        <item value="Ew" summary="Electric field W Component sampled at 35 kHz" />
      </items>
    </enum>

                                    (More lines follow...)
```

Note the highlighted selector names above. It is these names that will form the keys for the data queries.

## A.3: Das 2.3 Query Strings

TODO: write more...

# Appendix B: Das 2.3 Server – Reader Interface

*Das 2.1 is the current stable version. The basic interface and tools have been in use for over 10 years and work well for time series data. Das 2.2 is an expansion of this interface to handle arbitrary data indices and to make more meta-data available to clients.*

Das 2.1 readers only support connection-less operations. The reader program starts, answers reads it's command line to determine what data are to be returned, and then exists Das 2.2 supports two connection modes:

1. **Connection-less readers** – These readers take in a query request and produce an output stream in of the Das stream formats. No state data is maintained between requests.
2. **Bi-directional Readers –** These readers maintain an open channel between a client and a server.

## B.1: Data Source Interface Definition (DSID) Files

Das 2.2 servers use Data Source Interface Definition (DSID) files to understand how to advertise and control readers. DSID files must conform to the XML schema at:

```
http://www-pw.physics.uiowa.edu/das2/das_dsid-0.2.xsd
```

Example DSIDs are included at:

```
http://saturn.physics.uiowa.edu/svn/das2/das2Server/trunk/doc
```

A minimal DSID file follows, with main XML element names in bold. This particular example is for Voyager PWS low-rate data:

```
<?xml version="1.0"?>

<dasDSID xmlns="http://www.das2.org/dsid/0.2" name="Vgr1SAFull">

  <summary>Voyager 1 Spectrum Analyzer Full Resolution</summary>

  <description>
    Electric Field averages and peaks from the Voyager 1 PWS 16-Channel
    Spectrum Analyzer
  </description>

  <reader output="das2stream">
    <externalProcess interfaceVersion="2">
      <exec>/opt/project/voyager/Linux.x86_64/bin/vgpw_sa_rdr 1</exec>
    </externalProcess>
  </reader>

  <selectors>
    <range key="scet" name="SCET" format="DATETIME">
      <summary>Spacecraft Event Time</summary>
      <allow op=".beg."/> <allow op=".end."/>
    </range>
  </selectors>

  <output>
    <dimension name="SCET" quantity="event time" unit="utc">
      SCET of the point when the instrument cycle started.
    </dimension>
    <dimension name="Channel Center" quantity="frequency" unit="Hz">
```

```
      Center frequencies of each spectrum analyzer channel.
    </dimension>
    <dimension name="Electric Field" quantity="rms electric field" unit="V m**-1">
      The band limited RMS electric field difference across Voyager's electric
      antennas.
    </dimension>
  </output>

  <maintainer name="Chris Piker" email="chris-piker@uiowa.edu" />

 </dasDSID>
```

Each DSID file always has the following elements:

**dasDSID** – The top level element, provides the name of the data source and provides the XML name space for the elements.

**summary** – A one-line summary of the data source. This summary may be included in web-pages or other auto-generated data look-up tools

**description** – A longer description of the data source. This is a good place to put any caveats or processing notes.

**reader** – This block defines for the Das server how to run the reader. For the data source above the Das server is to run the program vgpw_sa_rdr, which generates a das2 stream, and send it's standard output to the client.

**selectors** – Data sources may support any number of selectors, but for this reader the only way to select data is by specifying a spacecraft event time range. The keyword for this selector is “sect”.

**output** – It's important to define the output of a reader so that client programs can know if data from this source may be combined on plots with data from other sources. Thus the output dimensions are specified here

**maintainer** – The last required parameter is the maintainer and their e-mail address. Problems inevitably arise and when they do it's important for testing software to know who to contact about the issue.

In addition to these elements others are supported, for a full reference see Appendix A.

## B.2: Das 2.3 Java Plugin Readers

Java objects which support the `org.das2.reader.Reader` interface may be run directly by a Das2.2 server. In order to locate and run a plugin reader an appropriate DSID file must be supplied.

## B.3: Command Line Readers

A Das server can run external programs to generate output streams. This section defines the command line call interface between a version 2.2 complaint Das reader and a Das server.

### *B.3.1: Return Value*

Readers shall return 0 if the request was successfully parsed and the appropriate data for that request were output. Note, the request my cover a range for which there were no data. This is a valid condition and 0 shall still be returned.

Readers shall return a non-zero value for any errors detected in the request itself, or errors in reading the input data, or errors in writing the output stream.

### *B.3.2: Command line Arguments*

Arguments provided to the reader consist of UTF-8 text strings. There are two types of arguments, *query parameters* and *reader directives*. The type of argument can be detected via the presence of an equals '=', sign (byte value 0x3D) in the argument. Query parameters **always** contain an equals character, reader directives **never**

contain an equals character.

#### DIRECTIVES

These values never contain a space or equals sign and are always provided as strings with explicit capitalization. Only one value is defined so far:

`keepalive`

When receiving the `keepalive` argument a reader that supports connected operations shall not exit, but shall remain running, waiting for control directives from Standard input. If a reader does not support `keepalive`, and this control value is received, then the reader must exit with a not zero return and sent a message to standard output noting that keepalive is not supported.

#### QUERY PARAMETERS

Das servers transmit data selection parameters to readers in the form of 'keyword operation value' triplets. Each triplet shall appear to the reader as a single input parameter. For example if the server runs a reader as follows:

*some_reader* `scet.beg.2012-02-01 scet.end.2012-02-02`

From the perspective of a reader implementer, the command line arguments are:

arg 0: `some_reader`

arg 1: `scet.beg.2012-02-01`

arg 2: `scet.end.2012-02-02`

The keyword section of the pair may not contain spaces, the value section may contain any UTF-8 sequence. There is no space between the '=' sign and the keyword or value. Note that since the value may contain any UTF-8 byte sequence, spaces are allowed in the value. Thus any space *after* the equals sign is part of the value.

The order of the pairs shall not affect the operation of the reader.

### *B.3.3: Relation between selectors and command line arguments*

DSID files define how to select data from a data source via XML `<selector>` elements. Here's the relationship between command line parameters and selector elements.

#### PARAM SELECTOR

This is the most general type of selector. It represents a continuous or roughly continuous parameter. Time, or more specifically, Spacecraft Event Time, is the most commonly supported selection parameter. Almost all Das2 readers support selecting data by time range. The snippet below compares the XML selector specification to some of the possible command line argument sets.

```
DSID:    <param keyprefix="scet" name="SCET" format="DATETIME" >
cmdline: scet.beg.2012-01-02  scet.end.2012-01-03
cmdline: scet.gt.2013-10-09T19:00
cmdline: scet.beg.2012-01-02  scet.end.2012-01-03 scet.ne.2013-01-02T19:00
```

Note the lack of an end time in the second command line example. This is legal and means that the data source should send all available data greater than the given start time. The comparisons `.beg.` and `.end.` are synonyms for `.ge.` and `.lt.` respectively.

Though the last command line example is perfectly legal most readers will not want to support a “not equals” comparison. The `<allow>` sub element may be used to prevent a reader from receiving unwanted comparison operations on the command line. The next example only the “equals” comparison is allowed.

```
DSID:<value key="fftlen" name="FFT Length" format="INTEGER" >
       <allow op=".eq" />
```

cmdline: fftlen`.eq.1024`

Multiple `<allow>` elements my be specified if needed.

#### ENUM SELECTOR

A single parameter is needed. The value portion of the command line argument comes from item's 'value' attribute:

```
DSID:<enum key="chan" name="Engineering Channel">
         <item name="+1.5V HFR Rail" value="hfr_1.5"/>
         ...
     </enum>
```

cmdline: `chan.eq.hfr_1.5`

#### BOOLEAN SELECTOR

A single parameter is needed, the value portion is any one item from the set {`true`, `false`, `1`, `0`} for example:

DSID: `<boolean key="rm_sndr" name="Remove Sounder">`

cmdline: `rm_sndr.eq.true`

### *B.3.4: Standard Input*

Das Readers which indicate in their DSID that they support `keepalive` must read standard input while delivering data. This is required to support control messages for data streams over BEEP. (See http://beepcore.org) The format of the input control messages is TBD.

Keepalive support is indicated in the DSID by adding the

`keepalive="true"`

attribute to the `<reader>` and `<reducer>` elements.

### *B.3.5: Standard Output*

Das2.2 Readers output data on the Standard Output file descriptor. The format must match that given in the associated DSID. Valid formats are Das2Streams QStreams or others not yet defined.

### *B.3.6: Standard Error*

Das2.2 Readers output error and status messages to the Standard Error file descriptor.

(TBD: The format of these messages is not yet defined. I'm assuming it's simply unformatted UTF-8 text.)

# Appendix C: DSID Schema Reference

The authoritative source for data source ID files is the schema document. The current schema file, as of August 12th, 2013 is `das_dsid-2.2.xsd`. Any information in this appendix that conflicts with the schema definition file is incorrect. That said, a textual overview is handy before attempting to read the XML definition directly.

# Appendix D: Proposed QStream Changes

Various proposals that are being considered.

**Length in packets, pipes used instead of colons.**

`|nn|dddddd` for packets. This would allow variable length packets, and gets rid of the problem when :01: appears in timetags.

### Enumeration Units encoding.

Presently all the values of an enumeration must be in the packet descriptor, which is tedious, limiting, and burdensome to programs. For example, programs must enumerate all the possible responses to define the unit. The das2 unit can have values added continually, and this should be available to the stream as well. For example,

<enum unit='unitId' value=2 text='Spacecraft spin flip'>

might allocate the number 2 to mean the message, and it would be up to the client to convey this information.

### Exceptions allowed without stream header.

The current parser allows the stream to be encoded without the header when an exception is thrown. This means that some valid streams start with [xx].

# Appendix E: Example Das2 Stream Header Packets

In the spirit of "An example is worth a 1000 words", here's a few examples

## E.1: Correlated time series (x multi y)

The following example is taken from the Mars Express MARSIS digitized density data reader. The records have three planes, <x> <y> <y>. It is an example of an X with multiple Y values.

```
[00]000158<stream>
  <properties
    String:xLabel="SCET (UTC)" Datum:xTagWidth="8.0 s"
    String:title="MARSIS Plasma and Magnetic Field Parameters"
  />
</stream>
[01]000413<packet>
  <x type="time24" units="us2000"></x>
  <y type="ascii11" name="dens" units="cm**-3">
    <properties
      String:yLabel="N!De!N" String:yDisplayName="Electron Plasma Density"
      String:summary="None"
    />
  </y>
  <y type="ascii11" name="mag" units="nT">
    <properties
      String:yLabel="B!Dmag!N" String:yDisplayName="B-Field Magnitude"
      String:summary="None"
    />
  </y>
</packet>
:01:2005-08-04T22:06:58.389   5.52e+02   2.05e+01
:01:2005-08-04T22:07:05.932   5.21e+02   2.09e+01
:01:2005-08-04T22:07:13.475   6.91e+02   2.06e+01
:01:2005-08-04T22:07:21.018   1.36e+03   2.01e+01
:01:2005-08-04T22:07:28.561   6.79e+02   2.26e+01
```