Das2

Client-Server Interface Control Document

Version 2.3 - Pre-release for Comments

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
Struck original appendix A and B, in prep for Version 2.3 ICD	C. Piker	2018-11-29
Das2-pyserver specific reference moved to pyserver user's guide	C. Piker	2019-08-20
New version, 2.3. See revision notes in Section 1.7.	C. Piker	2019-08-30
Version 2.3 revisions after internal comments, posted for public comment	C. Piker	2019-09-03

Recent Revisions

Table of Contents

Acronym	Meaning
CDF	Common Data Format, a data format defined by the Heliophysics division of NASA
EPN-TAP	Euro-PlaNet Table Access Protocol. Specific query protocol to access planetary science data.
GET	Not an acronym, but looks like one. An HTTP/1.1 transaction method.
HAPI	The Heliophysics Application Programming Interface, a single resolution stream server definition.
HTTP	HyperText Transfer Protocol, The basic transport protocol for most of the world's data.
ICD	Interface Control Document
IDL	Interactive Data Language, a proprietary programming language from Harris Geospatial Inc.
JSON	Javascript Object Notation, a text based machine to machine data object serialization standard
MIME	Multipurpose Internet Mail Extensions, an Internet standard for identifying and encoding information
PDS	The Planetary Data System, a standards archiving organization within NASA.
XML	eXtensible Markup Language
UTF-8	A multibyte character encoding for all the world's languages that is compatible older encodings

Acronyms used within this document

Foreword

As of August 2019 das2 is now 17 years old. The authors of das2 are well aware that much of the client-server interface codified in this document was not thoughtfully designed, instead it "just grew" to satisfy the needs of GUI clients that were in rapid development at the time. This interface definition, version 2.3, is intended to be the last in the 2.X series. Though the basic operating principles will remain the same, we look forward to refactoring the client-server interface to provide a more flexible, consistent and robust definition that will better serve the needs of the space physics community going forward.

Das2 developers include:

- Larry Granroth - Conceived the basic design of das classic in 1996 and later hired the initial development team for das2 consisting of Jeremy Faden and Edward West.
- Jeremy Faden - Initial das2 Java GUI developer and libdas2 developer, creator of the Voyager das2 client application and creator of Autoplot.
- Edward West - Initial das2 Java GUI developer and creator of multiple Cassini and Mars Express das2 client applications.
- Chris Piker - Rewrote libdas2, created das2py, das2-pyserver and the ICDs, along with many server side data readers.

...along with many others who have written readers to transmit their data via a das2 server.

You're welcome to join in future development efforts hosted at das2.org and github.com/das-developers.

When you notice errors or omissions in this document, you are welcome to notify the developers via e-mail at:

das-developers@uiowa.edu

or post issues to the https://github.com/das-developers/das2docs issue tracker.

1 Overview

This interface control document defines the interaction protocols and conventions for the *das2, version 2.3* 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 from the Plasma and Radio Wave Science instrument on board Cassini. Over time access was added for data from Voyager, Galileo, Polar, Cluster, Juno, Van Allen Probes, and other missions. In 2016 das2 "jumped the pond" and found use assisting data delivery for the MASER project at the Paris Observatory and in Summer 2019 das2 servers entered test runs at the Southwest Research Institute and the Czech Academy of Sciences.

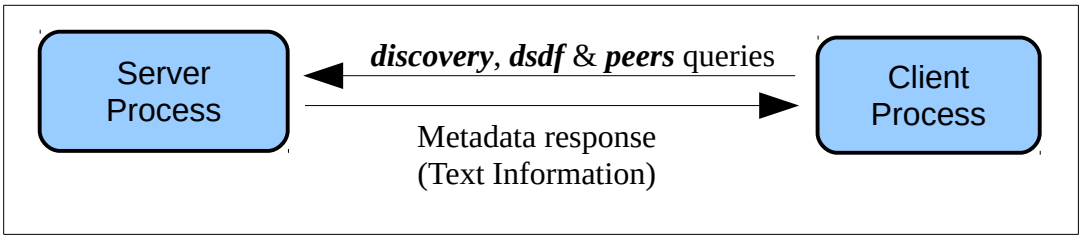
This documents sets forth the interface definition that allows any software to claim the name das2. If it supplies or consumes data via this interface, it can claim to be a das2 program regardless of where it's developed or what programming language is employed.

1.1 The Das2 System

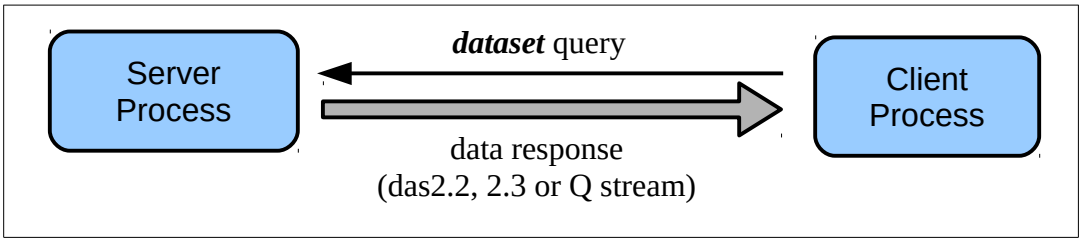
The das2 system as defined within this ICD consists of two basic parts.

1. One or more **servers**. These programs listen for HTTP GET queries and respond
2. One or more **clients**. These programs are the human interface. They request data and usually plot it.

Clients may request general information about the server, or one of more of it's data sources. These are termed ***discovery*** queries. From the list of sources, clients may request more information on a particular source, these interactions are call ***DSDF*** queries. The basic information flow is diagrammed below.



Metadata Queries



Data Retrieval Query

Using information from the ***DSDF*** query, clients may then 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

The first das2 server, www-pw.physics.uiowa.edu, ran on the Sun Solaris operating system and entered service in 2002 providing access to data sets from Cassini, Voyager and other spacecraft for over 16 years. Though the

original server no longer functions, it's services have been moved to more capable hardware. When developing clients it's handy to have a list of servers to communicate with for testing purposes. Towards that end, table 1.1 provides a listing of public das2 servers known to the authors as of August 2019.

Root URL and Organization	Data Sources
http://ipecman.ufa.cas.cz/das/server Czech Academy of Sciences, Institute of Atmospheric Physics	Interface for sPECtral Matrix ANalysis (iPECMAN[1])
https://das2server.obs-nancay.fr/das2/server The Paris Observatory, Paris Astronomical Data Centre	Observations from the Nançay Decametric array along with NenuFAR datasets
http://voparis-das-maser.obspm.fr/das2/server The Paris Observatory, Paris Astronomical Data Centre	MASER[2] data sets from Cassini, Voyager and others.
http://planet.physics.uiowa.edu/das/das2Server The University of Iowa, Department of Physics & Astronomy	Space plasma and magnetic field data with spacecraft location for Cassini, Voyager, Mars Express, and others
https://emfisis.physics.uiowa.edu/das/server The University of Iowa, Department of Physics & Astronomy	Space plasma and magnetic field data with spacecraft location for the Van Allen Probes
http://jupiter.physics.uiowa.edu/das/server The University of Iowa, Department of Physics & Astronomy	Space plasma and magnetic field data along with spacecraft location for Galileo, and Juno
https://zeus.physics.uiowa.edu/das/server The University of Iowa, Department of Physics & Astronomy	Ground radio observations of Jupiter from the Long Wave Array

Table 1.1: Public Das2 Server URLs, August 2019

Depending on a static list of servers in an ICD is not a very useful way to find das2 data sources. The Virtual European Solar and Planetary Access (VESPA) project at the Paris Observatory have undertaken efforts to list das2 data sources in EPN-TAP databases. A web tool is available for searching for das2 and other data sources at http://vespa.obspm.fr/planetary/data/. In addition a das2 federated catalog root-node has been setup at das2.org and mirrored at github.com. A web-based browsing tool is available https://das2.org/browse for viewing das2 and other data sources in the catalog.

1.3 Das2 Clients

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, available from http://autoplot.org/, provides an interface for browsing das2 servers, retrieving data, and interacting with the resulting plots and contains a built in python scripting engine.

Though Autoplot is the primary das2 client, there are others. As of August 2019, Southwest Research Institute's **SDDAS** suite, available from http://www.sddas.org/ is in the process of adding das2 support. The **das2pro** IDL module can be used to gather das2 streams into IDL, it's available from https://github.com/das-developers/das2pro. If you are working in the Anaconda Python 3 environment, the **das2py** conda module is available at https://anaconda.org/DasDevelopers/das2py to assist with generating matplotlib plots from das2 servers and for interacting with the das2 federated catalog.

1.4 Data Stream Formats

Currently two data stream formats are defined within this ICD, both three are considered stream formats in that client programs may begin processing data before the transmission completes. Data definitions always proceed data values.

das2 streams – These are self-describing data streams which contain headers to define the layout of succeeding

1 iPECMAN: http://ipecman.ufa.cas.cz/
2 MASER: https://github.com/maserlib

bytes and transmit data across all arrays a single slice at a time. This format supports a handful of fixed data value arrangements. Parsers exist for this stream format in Java, C, Python and IDL.

QStreams – These are also self-describing streams which headers and slice oriented transmission. This format is more general than das2 streams, supporting a wider variety of data value arrangements, though as of August 2019, it is only readable by Autoplot.

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 names *das1* and *das-classic*. 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. Though libdas2 provides utilities to up-convert das1 streams to das2 streams, continued development of das1 readers is not recommended.

1.6 HAPI

HAPI stands for "Heliophysics Application Programming Interface". It is an stream server interface definition inspired by das2 and other efforts. HAPI development was funded by Goddard Spaceflight Center to provide a minimal stream server definition. Similar to das1, HAPI streams are homogeneous.

On one hand, HAPI queries are a bit rigid, disallowing many operations that the creators of das2 considered essential for fluid data set navigation. On the other hand, HAPI data streams are relatively easy to parse and all metadata responses are JSON documents, which is a very convenient format. The HAPI is defined in it's ICD at https://github.com/hapi-server/data-specification. Though outside the scope of this document, some das2 programs such as das2-pyserver and Autoplot also support the HAPI.

1.7 Changes from Version 2.2

All references to a particular das2 server implementation, notably the das2-pyserver, have been removed to create a clean ICD usable by any server implementation. All expository text has been rewritten with the benefit of six years of hindsight since the original version 2.2 release. In addition, a few client-server protocol changes have been made for this version. These are listed below with relevant section numbers.

- The `interval` HTTP GET query parameter has been deprecated, it's now considered a synonym for `resolution` (section 2.3).
- A new array definition element, `<xscan>`, has been added to handle fixed X-axis offsets from varying X-axis reference points (sections 3.4.1 and 3.4.7).
- The `yTags` attribute of the `<yscan>` element has been deprecated and is considered a synonym for `yOffsets` (section 3.4.7).
- The new `xOffsets` attribute of `<xscan>` is permitted in `<yscan>` elements as well (section 3.4.7).

All protocol changes are marked in the text using a this light yellow background along with the small text annotation "{ver2.3}".

2 Das2.3 HTTP GET Interface

The purpose of a das2 server is to listen for queries and provide responses in a standardized format. The internal details of the server are not important so long as it adheres to the query interface defined in this section, and produces data as either a either das2 stream (section 3) or QStream (section 4). Also the internal details of the client, be it an interactive GUI program or a small plotting script, are not significant to this interface definition.

All client-server interactions consists of HTTP GET requests sent by the client program, followed by an HTTP response whose message body that contains the response data.[3] It is assumed in these examples that das2 servers support the HTTP/1.0 protocol, or higher. To nail it down, requests have the following syntax:

Generic Das 2 Request Pattern

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

Where *RESOURCE*, *QUERYSTRING* and *SERVER* are replaceable parameters. Note that as these are HTTP requests, each line should be terminated by the carriage return and newline characters (ASCII 0x0D 0x0A pair). In addition, data requests are HTTP message headers and thus *end with a single blank line*.

2.1 Discovery Query

This request provides a list of all data sources known to the server. To start the request, a TCP/IP connection is made to the das2 server. UTF-8 text similar to the example below is transmitted, and the connection is held open awaiting a response from the server. The lone query parameter needed for discovery requests and it's fixed value are given below.

Query Key	Value Description	Required
server	Must contain the string: `discovery`	**yes**

The following example text, if encoded as UTF-8 bytes, could be sent to a server (by telnet if you wish) in order retrieve a list of data sources provided by the server.

Discovery Request Example

```
GET /das2/das2Server?server=discovery HTTP/1.1
Host: planet.physics.uiowa.edu
Accept: *
```

Discovery Response Example

```
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/|Saturn System Explorer - Orbit Insertion 2004-06-30
Cassini/Ephemeris/|Spacecraft location data - Jet Propulsion Laboratory (NAIF)
Cassini/Ephemeris/Dione|Dione centered Cassini orbit parameters
Cassini/Ephemeris/Dione_CoRotation|Dione centered Cassini location - Cartesian Co-Rotational
Cassini/Ephemeris/Enceladus|Enceladus centered Cassini orbit parameters
```

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

The text above demonstrates a legal response from the server, which would then close the TCP/IP connection. The return value is new-line delimited. Each line contains the following record structure:

```
KEY | Descriptive Text
```

The KEY and Descriptive Text are separated by a single pipe, (i.e. **`|`**), character. Keys may be used to define directories, which are just categories of sources, or to define sources which can provide data. Keys that end in "**`/`**" are directories and may be ignored by clients that only present a flat name space. Each key is non empty and has one of the following formats.

```
DIRECTORY/ [ DIRECTORY/ ...]
DIRECTORY/ [ DIRECTORY/ ...] SOURCE
```

Items in brackets "[ ]" are optional. Items in brackets followed by ellipses may repeat. Keys may not contain spaces and Source Keys may not end in a forward slash (i.e **`/`**).

All keys define a callable data source that can be contacted using a ***`DSDF`*** query.

2.2 DSDF Query

This request provides information on a particular data source, presumably found by a *`discovery`* query. *`DSDF`* queries require the following HTTP GET query parameters.

Query Key	Value Description	Required
`server`	Must contain the string: `dsdf`	**yes**
`dataset`	The data source to query, this would be one of the key values returned from a *`discovery`* query.	**yes**

In the following example, a client gathers more information on a Juno Waves high-rate data source.

Source Request Example

```
GET /das2/server?server=dsdf&dataset=Juno/WAV/Burst_Waveform HTTP/1.1
Host: jupiter.physics.uiowa.edu
Accept: *
```

This returns a das2 stream header providing information on the data source. The example below contains both required and optional properties attributes, which is typical of most server responses. The **required** properties are in **bold**. Extra properties are okay. An important principle of the *`DSDF`* response interface is that:

> *Client programs should **ignore** properties that they do not understand.*

Following this principle allows for flexibility to address future needs without breaking compatibility.

Source Response Example

```
HTTP/1.1 200 OK
Date: Tue, 23 Jul 2019 21:21:55 GMT
Server: Apache/2.0.63
Expires: Tue, 23 Jul 2019 21:21:55 GMT
Connection: close
Content-Type: text/vnd.das2.das2stream; charset=utf-8

[00]000738<stream>
  <properties
    description="Juno Waves, calibrated directly-sampled waveforms"
    das2Stream="1"

    techContact="Chris Piker &lt;chris-piker@uiowa.edu&gt;"
    sciContact="Bill Kurth &lt;william-kurth@uiowa.edu&gt;"
```

Source Response Example (continued)

```
    exampleRange_00="2013-10-09 to 2013-10-10 | Earth Flyby"
    exampleRange_01="2016-240T6:50 to 2016-240T18:50 | Perijove 1"
    exampleRange_02="2017-086T02:00 to 2017-086T16:00 | Perijove 5"

    param_00="LFR_LO_E | Electric field waveforms sampled at 50 kHz default"
    param_01="LFR_HI_E | Electric field waveforms sampled at 375 kHz"
    param_02="HFR_BASE | Electric field waveforms sampled at 7 MHz"
  />
</stream>
```

The authors of this ICD are aware that reusing das2 stream headers for this purpose is a bit odd. However, more robust server independent data source advertisement mechanisms are available though das2 federated catalog nodes or EPN-TAP database entries so this response type has not been changed for this version of the interface definition.

2.2.1 DSDF Response XML Attribute Tables

The tables below list property attributes that are returned in response to a *DSDF* query. Note that some properties end with the characters "**XX**". This is a replaceable token indicating a two-digit, zero padded number in the range of 00 to 99. In some cases, such as **exampleRange_XX**, **exampleInterval_XX** and **exampleParams_XX** the two-digit number is used to match entries that are used together when forming *dataset* queries.

The following property element attributes must always be provided by the server in all cases.

Properties Attribute	Description
description	A title for the source
das2Stream/qstream	One of these must exist and have a value of "1" depending on the *dataset* query output format.
exampleRange_XX	Here XX is a two-digit zero padded number from 00 to 99. *At least one example time range must be provided*. The range is given as: **MIN_TIME to MAX_TIME UNITS | Descriptive Text** where the UNITS and Descriptive Text are optional.
techContact	Technical contacts who can resolve problems with the data source. Format is: name1 <email1> [, name2 <email2> ...]

Table 2.1: Required property attributes for all DSDF responses

The following property element attributes must exist depending on the given situation.

Properties Attribute	Description and Required Condition
requiresResolution {ver2.3}	If the source requires the **resolution** query parameter in its *dataset* queries, then this attribute will be present in the *DSDF* XML return and have the value "1".
requiresInterval	A synonym for requiresResolution
exampleResolution_XX	This is a companion to the exampleRange_XX property for interval based data sources, it provides the value of **interval** to use when asking for the corresponding example range in *dataset* queries. If requiresResolution is present and set to "1" then this parameter shall also be present.
exampleParam_XX	This is a companion to the corresponding exampleRange_XX property. If example XX requires a certain value of the **params** query parameter in *dataset* queries
reducible {ver2.3}	By default data are assumed to be reducible on the server by averaging over time bins. *Not all sources support this operation.* If the source ignores the **resolution** query parameter then this attribute should be present and set to "0".

Table 2.2: Property attributes conditionally required in DSDF responses

Optional properties that are commonly present, though not required.

Properties Attribute	Description and Required Condition
sciContact	One or more contact email addresses for questions about the scientific usefulness or interpretation of the data from this source. Values formated as **techContact** above.
summary	A long form string describing data processing and other details of the source. May included string formatting escapes as described in section 3.3.2.
usagePolicyUrl	A HTTP URL that provides conditions and acknowledgments for persons using this data source, especially in publications.
param_XX	In general the **params** value is undefined and varies from data source to data source, but for many data sources the **params** value can be treated as a white space separated flag set. Assuming that the contents of the **params** query parameter can be treated as a flag set, each attribute with this name pattern defines one of the flags and provides a description for it. The format is: `FLAG_VALUE [ | Descriptive Text ]` where [ ] indicates an optional item, so the descriptive text is optional. The formation of the complete **params** HTTP GET value would thus be: "`FLAG1 FLAG2 FLAG3`" etc., depending on the number of flags in the request.
item_XX	Provides information on a single dependent data array that will appear in the stream output by an associated *`dataset`* query. Used by some GUI tools to limit the data that are displayed.

Table 2.3: Common optional property attributes in `DSDF` responses

2.3 Dataset Query

In the das2 query protocol **all** data are assumed to be indexed by time, and **all** queries that return data require a starting and an ending time point. Though still relatively simple, *`dataset`* requests are the most complex queries defined by this ICD as time resolution and optional parameters are supported.

One of the central features of das2 is that data streams can be reduced in size by averaging values into time bins on the server before transmission to the client. This saves network bandwidth, and may shorten response times. Client programs state the desired upper-bound on the width of the time bins when communicating with the server by setting the **resolution** query parameter as described in in table 2.4 below.

Query Key	Value Description	Required
server	Must contain the string: `dataset`	**yes**
dataset	The data source to query, presumably provided by an earlier *`discovery`* query.	**yes**
start_time	Any parse-able time point format. The ISO-8601 time format is a subset of the allowed time formats.	**yes**
end_time	A time point greater than the value of `start_time`.	**yes**
resolution	The time bin size, in floating point seconds, for which data are preferred. By default this parameter is optional. If the `<properties>` attribute `requiresResolution` (or it's synonym `requiresInterval`) is set to "`1`" in the *`DSDF`* response, then this query parameter is required. The reason for this conditional behavior is explained in the expository text following this table. {ver2.3}	maybe
params	Extra information may be passed to a reader via this parameter.	no
`interval`	This parameter is deprecated. Servers should treat it as a synonym for resolution.	no
anykey	Das2 servers are free to allow other parameters outside this specification	no

Table 2.4: Das 2.3 `dataset` Query Parameters

Normally, it is assumed that there exists some inherent time scale in the original data and so the `resolution` parameter is optional. However, some sources may produce data from mathematical models and therefore have effectively infinite resolution. Since no intrinsic time scales exist for such sources, they need the client to give them one, and they advertise this fact in their *DSDF* responses.

Servers should produce data at the requested time resolution, *or finer*, unless the intrinsic time spacing of the data is longer than the requested resolution. The easy way for servers to satisfy this requirement would be to always send full resolution data, which is a bad idea, as this negates all the advantages of server side data reduction. Instead, servers should send data with time bins at, or slightly shorter than, the requested resolution if possible. If for some reason a source can not alter it's time resolution, and the client provides a `resolution` query parameter anyway, then the server should just ignore the resolution keyword.

The example query below uses the minimum number of data request parameters.

Data Request Example

```
GET /das/server?server=dataset&dataset=THEMIS/THEMIS_A&start_time=2008-06-
29T17:11&end_time=2008-06-29T17:12 HTTP/1.0&params=BxBx HTTP/1.1
Host: ipecman.ufa.cas.cz
Accept: *
```

For this particular data source, the response body contains a das2.3 stream. For more details on the das2.3 stream format, see section 3.

Edited Data Response Example (das2.3 stream)

```
HTTP/1.1 200 OK
Date: Wed, 28 Aug 2019 20:54:32 GMT
Server: Apache/2.4.25 (Debian)
Expires: now
Content-Disposition: attachment; filename="THEMIS_A_2008-06-29T17-11_2008-06-29T17-12.d2s"
Connection: close
Content-Type: application/vnd.das2.das2stream

[00]000145<stream version="2.3">
  <properties
     title="THEMIS-A : SCW-EFW, BxBx*"
     Generated_by="IAP CAS Prague (U. Taubenschuss)"
  />
</stream>
[01]000337<packet>
  <x type="little_endian_real8" units="t2000" />
  <yscan type="little_endian_real4" name="BxBx" nitems="164" zUnits="mV**2 m**-2 Hz**-1"
         yUnits="Hz" yOffsets="0.0e+00,2.5052e+01,5.0104e+01,7.5156e+01,1.0021e+02" >
    <properties yLabel="Freq (Hz)" zLabel="BxBx* (mV**2 m**-2 Hz**-1)" yScaleType="linear"/>
  </yscan>
</packet>
(Binary das2 Data Packets follow...)
```

2.3.1 Redirects

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 the server may (at it's option) issue an HTTP redirect instead of a HTTP 404 not found message. In the example below, the server `planet.physics.uiowa.edu` was requested to provide Juno Waves Survey data. Since the das2 server on planet happens to know that those data are provided by the host `jupiter.physics.uiowa.edu`, a redirect response is returned.

Redirect Response Example

```
HTTP/1.1 302 Found
Date: Thu, 25 Jul 2013 21:53:03 GMT
Server: Apache/2.0.63
Location: http://jupiter.physics.uiowa.edu/das/das2Server?server=dataset;dataset=Juno%2FWAV
%2FSurvey;start_time=2018-001;end_time=2018-002
Content-Length: 0
Connection: close
```

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

2.4 Restricted Dataset 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.

Restricted Dataset Request Example

```
GET /das2/server?server=dataset&dataset=Nancay/NDA/JunoN/junon_tag1_xa&start_time=2017-07-
01T17:14:00&end_time=2017-07-01T17:15:00&resolution=0.1 HTTP/1.1
Host: das2server.obs-nancay.fr
Accept: *
```

Authorization Required Response Example

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Basic realm="Nançay Decameter Array Team Access"
```

At this point the client program gathers a user name and password by what ever means it prefers, encodes it using the rules of HTTP Basic Authentication, and tries again.

Example Dataset Request with Authorization[4]

```
GET /das/server?server=dataset&dataset=Nancay/NDA/JunoN/junon_tag1_xa&start_time=2017-07-
01T17:14:00&end_time=2017-07-01T17:15:00&resolution=0.1 HTTP/1.1
Host: das2server.obs-nancay.fr
Authorization: Basic YnJoYF4tdG90dH76T3ch
Accept: *
```

Since the `Authorization` header is now included, assuming the token is valid, a normal *`dataset`* response is returned as described in section 2.3.

2.5 ID Query

Das2 servers can provide a small text string identifying the organization hosting the server, this string is retrieved via an *`id`* query.

Query Key	Value Description	Required
`server`	Must contain the string: `id`	**yes**

The response is a single line of UTF-8 text as the sole content of an HTTP message body. Servers are encouraged

4 The authorization token in this example is fake, of course

to provide different strings depending on the value of the:

`Accept-Language`

HTTP Header. An example of an *`ID`* query follows.

ID Request Example

```
GET /das/server?server=id
Host: emfisis.physics.uiowa.edu
Accept: *
```

The response body is a one line of UTF-8 text identifying the organization such as "The University of Iowa, Department of Physics and Astronomy".

2.6 Logo Query

Das2 servers may provide a small image to be used in server listings via the *`logo`* query.

Query Key	Value Description	Required
`server`	Must contain the string: `logo`	**yes**

An example of a logo query follows.

Logo Request Example

```
GET /das/server?server=logo
Host: emfisis.physics.uiowa.edu
Accept: *
```

The return value is a HTTP message with a PNG, or JPEG image as the sole content of the message body.

2.7 Peers Query

Das2 servers can (but are not required to) advertise the existence of other das2 servers. If a server supports providing a list of co-located or cooperating servers then a client program my request this list using the query syntax given in this section.

Query Key	Value Description	Required
`server`	Must contain the string: `peers`	**yes**

For now, the following example exchange *defines* a peers query and response, because as of August 2019, no XML schema definition file has been created for this response type. Das2 servers are free to alter the output of the response depending on the value of the:

`Accept-Language`

HTTP Header.

Peers Query Example

```
GET /das/das2Server?server=peers HTTP/1.1
Host: jupiter.physics.uiowa.edu
Accept: *
```

The associated response is returned.

Peers Response Example

```
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="/static/das2Server.xsl"?>
<das2server>
  <peers>
    <server>
      <name>zeus</name>
      <url>https://zeus.physics.uiowa.edu/das/server</url>
      <description>Hosts ground based radio-astronomy observations from LWA-1</description>
    </server>
    <server>
      <name>planet</name>
        <url>http://planet.physics.uiowa.edu/das/das2Server</url>
        <description>Outer planet mission data server</description>
    </server>
    <server>
      <name>emfisis</name>
      <url>https://emfisis.physics.uiowa.edu/das/server</url>
      <description>Van Allen Probes quicklook server</description>
    </server>
  </peers>
</das2server>
```

3 Das2.3 Stream Format

Typical space physics data sets have a much greater extent in the time dimension than in other independent physical dimensions. For example, the electric field spectrum analyzers on the Voyager missions only have 16 frequency channels, but a sweep has been collected about 4 times a minute for over 40 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. The basic time stream principles are:

(1) Increasing byte offsets correspond with increasing time values.
(2) A complete time slice across all arrays is delivered in each packet.

These concepts are illustrated in figure 3.1 to the right.

The section defines the das2.3 stream format, but before diving into the details, the reader should be aware up front that there are a couple major limitations of das2.3 streams:

Header packets only have definitions for arrays up to rank 2.

Das2.3 data packets are homogeneous, and thus can not handle ragged arrays, though fill values are acceptable.

The das2 developers are well aware of these, and other, limitations and intend to address them in the future edition.

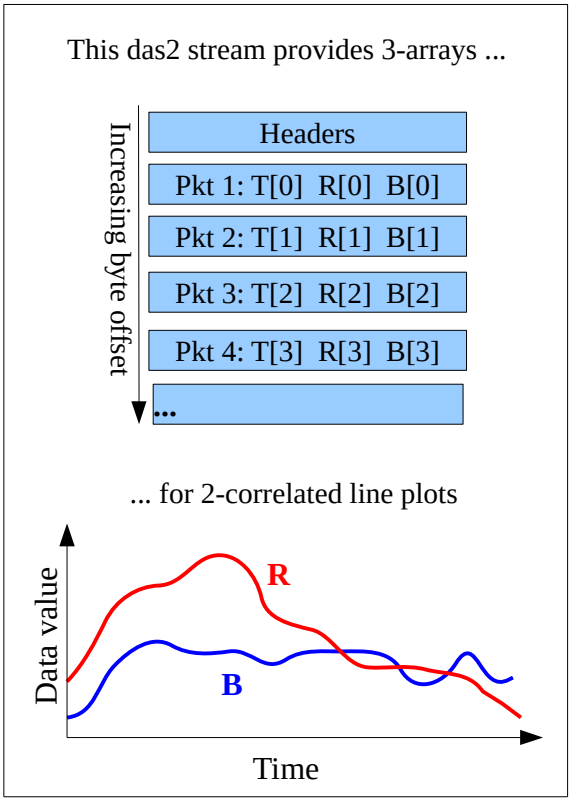


Figure 3.1: Stream versus Plot comparison

3.1 Stream Identification and Layout

3.1.1 MIME Types

All das2 streams may be identified by web-browsers, email applications and other programs using the following MIME Content-Type and file name pattern:

MIME: **`application/vnd.das2.das2stream`**
Extension: ***.d2s**

Das2 streams that happen to have all values encoded using the text formats **`asciiN`** and **`timeN`** (section 3.4.7), will consists entirely of UTF-8 text. For these streams, the following identification is recommended:

MIME: **`text/vnd.das2.das2stream; charset=utf-8`**
Extension: ***.d2t**

3.1.2 Layout

A das2 stream consists of solely of packets. There are four types of packets:

- *A Stream Header Packet:* There is only one of these and it is the first item in the stream.
- *Header Packets*: These contain array structure and value format definitions.
- *Data Packets*: These contain the actual data, typically in binary form.
- *Info Packets*: These contain messages such as stream progress and exceptions.

Each das2 header packet defines a group of arrays that are correlated in index space. The values for these arrays are carried by the corresponding data packets. Each data packet provides a complete set of all array values for a *single* value of the *slowest* moving index. Thus, as an example, if a header packet defined the following arrays:

```
X[*] Y[*] Z[*,6]
```

where * indicates an array index that can grow to any desired positive integer, then first data packet would contain the values:

X_0 Y_0 $Z_{0,0}$ $Z_{0,1}$ $Z_{0,2}$ $Z_{0,3}$ $Z_{0,4}$ $Z_{0,5}$

and the second would contain:

X_1 Y_1 $Z_{1,0}$ $Z_{1,1}$ $Z_{1,2}$ $Z_{1,3}$ $Z_{1,4}$ $Z_{1,5}$

and so on. This is a key feature that allows for processing any size stream without running out of memory.

Each Header Packet can define a different set of arrays. Header Packets are matched to corresponding Data Packets using Packet IDs. Most packet IDs are two-digit ASCII integers that range from **01** to **99**. ID **00** is used to denote the Stream Header Packet, and the non-numeric ID **xx** (i.e. two literal ASCII small x characters) is used for all info packets.

After the Stream Header packet, there must be at least one Header Packet. A Data Header of a given ID must occur in the stream somewhere before Data Packets of the same ID. There is no requirement to group Data Packets by type, packets may be interleaved as illustrated below.

```
[00]<stream> ... </stream>
[10]<packet> ... </packet>
:10: data values for major index 0 of packet type 10
:10: data values for major index 1 of packet type 10
[27]<packet> ... </packet>
:10: data values for major index 2 of packet type 10
:27: data values for major index 0 of packet type 27
:27: data values for major index 1 of packet type 27
:10: data values for major index 3 of packet type 10
...
```

Info Packets can appear at any location in the stream *other than the first packet*. The Stream Header Packet is **always** first. The next sections describes packet formatting details.

3.2 Packet Prefixes

Each packet is prefixed by a small tag that contains it's **Packet ID**, and for non-data packets, it's length.

3.2.1 Non-data packets

All *non-data* 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 i.e. **[**	Packet ID, zero padded UTF-8 integer, or "**xx**"	0x5D, i.e. **]**	Packet length in bytes, zero padded UTF-8 integer	Packet Contents

Note that the packet length is the number of *bytes* in the content, which is not necessarily the same as the number of *characters* since non-data packets contain UTF-8 encoded text.

3.2.2 Data packets

All data packets have a four byte prefix consisting of UTF-8 text characters. Encoded in the prefix is the Packet ID, however the length is not specified. The length of data packets is determined by parsing the associated Header Packet contents. Though details are provided in section 3.4.7, in short, the length of a data packet is:

The sum of:

the bytes needed to encode a single value of a data array

times

the number of items per packet for the array (which is 1 for one-dimensional arrays),

for all arrays defined in the associated Header Packet.

The table below details the contents of the data packet prefix.

Byte Index:	0	1-2	3	4 to Packet Length + 4
Value:	0x3A i.e. **:**	Packet ID, zero padded ASCII integer	0x3A i.e. **:**	Data Packet length determined by parsing associated Header Packet

3.3 Stream Header Packet

Every das2 stream must start with a Stream Header Packet. Stream headers always have packet id "`00`". The content of a the packet is an XML message with the top level element **`<stream>`**. The <stream> element may have a single <properties> child, though this is not required. The character encoding for stream headers is UTF-8. An example packet follows.

```
[00]000178<stream version="2.3">
  <properties
    Datum:xTagWidth="128.000000 s"
    double:zFill="-1.0e+31"
    String:title="Cassini RPWS Electric Survey, Key Parameters"
  />
</stream>
```

The following table defines the XML elements that make up a stream header packet. The general format of the attributes in the `<properties>` element are:

`TYPE:NAME = "VALUE"`

For properties of TYPE "**`String`**" the TYPE and it's following colon may be omitted.

Das2 was designed before XML namespaces were common, and thus namespace aware XML parsers will often fail if given the contents of a das2 header. This will be corrected in a future version of the stream format.

3.3.1 Stream and Properties Attribute Table

The stream element has a single required attribute, the version number.

Attributes of <stream>	
`version` (required)	For streams conforming to this document the version attribute value should always be `"2.3"`.

In addition, the `stream` attribute may have one sub-element named `properties`. The `properties` sub-element of `stream` is the primary meta-data location for the stream as a whole. Any attribute is allowed in properties, though some of these will have special meanings to das2 clients and server-side tools as described below. Attributes of properties are defined in the Table 3.1 below.

The authors of this standard are aware of the awkwardness of combining both the *datatype* and *purpose* of a value into the attribute name in a way that suggests XML namespaces are in use, when they are not. This version of the interface has been left as is to avoid too many changes for stream parsers.

Attributes of <stream> <properties>	
String:renderer	Used to suggest a plot type to the client program, values may be 'spectrogram', 'symbolLine', 'stackedHistogram'.
String:title	This provides a title for plots generated from this stream
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 the das2 server must be contacted for new data after a zoom or pan operation, or if the data already provided are sufficient. Proper use of xCacheRange and xCacheResolution tags can increase the performance of interactive das2 clients.
Datum: xCacheResolution	Used with xCacheRange, this attribute provides the resolution of a stream in the X axis direction. This attribute should always be set for data sources with a known resolution, (i.e. spacecraft ephemerides, or time bin reduced streams). If this attribute not present, the data are considered to be at intrinsic resolution.
String:xFormat	Provides a C-like format for text display of X-axis values. See section 3.3.3.
String:xLabel	Provides an X-axis label for plots generated from this source. Labels may contain formatting flags which allow for subscripts and superscripts, see section 3.3.2 for details.
boolean: xMonotonic	An important attribute that if set to "true" states that all X array values within each packet type are sorted in increasing order. Packet types themselves may occur in any order.
DatumRange:xRange	Useful for setting the X scale in client programs. The format of the attribute value is "*START* **to** *END UNITS*", where *START* and *END* are the inclusive beginning and **exclusive** ending values, and *UNITS* is the units string. Note that the UNITS portion can be omitted if START and END are ISO-8601 time strings. The range may be reversed as a hint to client plotting progarms.
String:xSummary	A description of the X axis values. This information is not intended to be used as a plot label and can thus be a few lines of text. Line breaks and other formatting flags defined in section 3.3.2
Datum:xTagWidth	Defines the minimum gap in the X direction across which client plotting programs should **not** interpolate. A change smaller then this value may be interpolated via connecting lines or other visual interpolation methods.
double:xValidMax	Provides a maximum possible valid data value for X axis values. Data above this value should not be displayed.
double:xValidMin	Provides a minimum possible valid data value for X axis values. Data below this value should not be displayed.
double:yFill	Specifies a fill value for missing Y data values. Defaults to -1e31 if not specified. Most clients will ignore yFill when <yscan> and <z> arrays are present in the packet definition (see Packet Headers 3.4 below.)
String:yFormat	Provides a C-like format for text display of Y-axis values. See section 3.3.3.

Attributes of <stream> <properties>	
String:yLabel	Provides an y-axis label for plots generated from this stream, may use formatting flags defined in section 3.3.2.
DatumRange:yRange	Useful for setting the Y scale in client programs. See the `DatumRange:xRange` attribute for formatting details
String:yScaleType	May be one of the strings "`log`" or "`linear`"
String:ySummary	A description of the Y axis values. This information is not intended to be used as a plot label and can thus be a few lines of text. Line breaks and other formatting flags defined in section 3.3.2
Datum:yTagWidth	Defines the maximum gap in the Y direction across which client programs should **not** interpolate. A change smaller then this value may be interpolated via connecting lines or other visual interpolation methods.
double:yValidMin	Provides a minimum possible valid data value for Y axis values. Data below this value should not be displayed.
double:yValidMax	Provides a maximum possible valid data value for Y axis values. Data above this value should not be displayed.
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.
String:zFormat	Provides a C-like format for text display of Z-axis values. See section 3.3.3.
String:zLabel	If present provides a z-axis label for plots generate from this dataset.
DatumRange:zRange	Useful for setting the Z scale in client programs. See the `DatumRange:xRange` attribute for formatting details
String:zScaleType	May be one of the strings "log" or "linear"
String:zSummary	A description of the Z axis (colorbar) values. This information is not intended to be used as a plot label and can thus be a few lines of text. Line breaks and other formatting flags defined in section 3.3.2
double:zValidMin	Provides a minimum possible valid data value for Z axis values. Data below this value should not bet displayed.
double:zValidMax	If present provides a maximum possible valid data value for Z axis values. Data above this value should not be displayed.

boolean:*AnyID*
String:*AnyID*
double:*AnyID*
Datum:*AnyID*
DatumRange:*AnyID*
int:AnyID
	In addition to the named items below, generic metadata may be provided, though client programs will not necessarily know what to do with these extra properties. Examples of using this feature:
	
	`String:maintainer="some.person@uiowa.edu"`
	`Time:creation_time="2014-04-23T00:30:23"`

Table 3.1: Das2.3 Properties Attributes

3.3.2 Label and Title 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. These escape sequences were inspired by IDL[5] but clients are only expected to support the following subset.

5 IDL Embedded Formatting Commands: https://www.harrisgeospatial.com/docs/embedded_formatting_comm.html

!A shift up one half line
!B shift down one half line
!C newline
!D subscript 0.62 of old font size.
!U superscript of 0.62 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, for example, "`!A`" and "`!a`" must have the same effect in a das2 client program. Characters that are not part of an understood format specifier are to be repeated in the output.

3.3.3 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` provide hints on how data should be printed as ASCII strings. The following C-style conversion specifiers should (but are not required to) be supported by das2 clients:

`%.Nf` - Decimal format, where N is an ASCII integer giving the precision after the decimal

`%.Ne` - Exponential format, where N is an ASCII integer giving the precision after the decimal

3.3.4 Macro Substitution in Attribute Values

Das2.3 stream data may pass through multiple programs before being read by a client program. Some of the keywords in the stream, such as the xCacheResolution may be updated by various programs as needed along the way. To make processing information more transparent to end users, macro expansions may be added to string attributes in the stream headers. Any value may be substituted so long as an attribute does not reference itself.

The following example header will help to illustrate the concept. 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 example was taken from the output of the standard libdas2 utility program `das2_bin_avgsec`, which automatically adds the %{xCacheResolution} macro to the stream header as the data pass through.

```
[00]000494<stream compression="none" version="2.3" >
  <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>
```

The format for das2.3 stream macros is:

```
"%{" + ANY_PROPERTY_NAME + "}"
```

There are a few items to remember when using attribute value macro substitutions:

1. The property TYPE is not used in the reference, just it's name.
2. Substitutions work for custom attributes, not just the standard items defined in section 3.3.
3. Since das2 stream attributes cascade (see section 3.4.7) the nearest attribute value is substituted.
4. Self referential macros are not allowed, i.e. don't use `%{title}` in a `title` attribute value.

This feature exists to support well formatted supplementary information in plot labels.

3.4 Header Packets

Barring any intervening Info Packets, the Stream Header Packet will be followed by at least one Data Header. The purpose of the Data Header is to define the arrays and value encodings for matching Data Packets. Data Headers and Data Packets are matched by the **Packet ID**. Header Packet contents are an XML document with the element `<packet>` as the root item. Header Packet contents are structured as follows:

```
<packet>
      <x>     <properties /> </x>          0 to N occurrences, properties optional
      <xscan> <properties /> </xscan>      0 to N occurrences, properties optional
      <y>     <properties /> </y>          0 to N occurrences, properties optional
      <yscan> <properties /> </yscan>      0 to N occurrences, properties optional
      <z>     <properties /> </z>          0 to N occurrences, properties optional
</packet>
```

Even though all sub-elements of `<packet>` are optional, *at least one* sub-element has to exist.

3.4.1 Array definitions in Header Packets

For conceptual simplicity, das2 arrays have been defined in terms of axes in a Cartesian space. This is just a simplification. Das2 client programs may display data along whatever coordinate axes are desired and may of course transform the data values in any manner they see fit. Though text formatting escapes are defined above, data plotting itself is outside the scope of this ICD.

The top level element, `<packet>`, has no attributes. It does have child elements.

Each child element of `<packet>` defines one array who's values will appear in Data Packets with a corresponding **Packet ID**. Array element definitions follow, and their attribute values are enumerated in section 3.4.7. When reading the array element definitions below it may help to refer to the complete `<packet>` examples provided in sections 3.4.2 through 3.4.5.

- **<x>** This child element of `<packet>` defines a rank-1 array that contains X-axis values. Each data packet will contain a single X-axis reference point from this array.

  Since the das2 server query protocol requires time boundaries, values for this array type are typically UTC times in some epoch, though this is *not required*. Client programs much check the `units` attribute of `<x>` instead of making assumptions.

  If an `<x>` array is not defined for a packet type then the X reference values will have dimensionless units and the value will correspond to the "slow moving index" i.e. the count of packets read, minus 1.

  Server implementers should be aware that most clients expect at least one `<x>` array to be defined.

- **<y>** This child element of <packet> defines a rank-1 array which contains Y-axis values. Each data packet will contain a single Y-axis reference point from this array.

  If there are no <y> arrays defined with a the packet element, then Y reference points are taken to be 0.0 for all corresponding Data Packets. If there is more than one <y> array in the packet definition, then each is considered to represent a *different* Y-axis.

- **<xscan>** A table element. This more complicated child of <packet> defines both a rank-1 X-offset array *directly with the element* via the xOffsets attribute, and a rank-2 array of Y-axis values that will appear in the corresponding data packet.

  To compute the X coordinates for each data value from this array, add the value of each xOffset to the value of <x> on a packet by packet basis. If more than one <x> array is present in the packet, use the first one. Thus the packet <x> value sets the reference point, and all the <xscan> values in a packet spread in X from that point.

  The number of Y values per packet for each <xscan> is given by its **nitems** attribute. The nitems attribute is thus also the size of the "fast moving index", or in C memory layout, the second index of the array. {ver2.3}

- **<z>** This child element of <packet> defines a rank-1 array that contains Z-axis (i.e. colorbar) reference points. Each data packet will contain a single Z-axis reference point from this array.

  There may be any number of <z> arrays, each one is considered to represent a *different* Z-axis.

- **<yscan>** A table element. Another complicated child of <packet> that defines both a rank-1 Y-offset array *directly within the element* itself via the yOffsets attribute, and a rank-2 Z-axis reference point array whose values are provided in the corresponding data packets.

  To compute the Y coordinate for each data value from this array, add the value of each yOffset to the value of the <y> array on a packet by packet basis. If no <y> array is defined, then Y reference values are taken to be 0.0. If more than one <y> array is present, use the first one for the reference. {ver2.3}

  The number of Z values per packet for each <yscan> is given by its **nitems** attribute. The nitems attribute is thus also the size of the "fast moving index", or in C memory layout, the second index of the array.

  Optionally, <yscan> may contain a rank-1 X-offset array directly within the element itself via the xOffsets attribute. If the xOffsets attribute is missing, the X-offsets are taken to be 0, if present X coordinates are calculated in the same manner as <xscan> above. {ver2.3}

To summarize, the following packet sub-element patterns are named, and well-defined ...

Short Name	Array Definition Pattern	Typical Use
X-Tag	<x> <y> **[** <y> <y> **...]**	Spacecraft location, mag vectors
X-Scan	<x> <xscan> **[** <xscan> <xscan> **...]**	Waveforms {ver2.3}
XY-Tag	<x> <y> <z> **[** <z> <z> **...]**	Derived results, scatter data
XY-Scan	<x> **[** <y> **]** <yscan> **[** <yscan> <yscan> **...]**	Fixed or variable freq. spectrograms

though others are not invalid.

3.4.2 X-Tag Example

The das2 stream fragment below is taken from the Van Allen Probes ephemeris data source. This is an X-Tag stream providing four time correlated arrays of probe A position information. 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.

```
[00]000142<stream version="2.3">
  <properties
     title="Van Allen Probe A - Location"
     xLabel="UTC"
     boolean:xMonotonic="true" />
</stream>
[01]000436<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="degrees">
    <properties String:yLabel="MLat" />
  </y>
  <y type="ascii11" name="mag_lt" units="degrees">
    <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
```

Each <y> array thus provides Y values in different units and thus would typically not be plotted on the same Y axis, though clients will of course collapse these as they see fit.

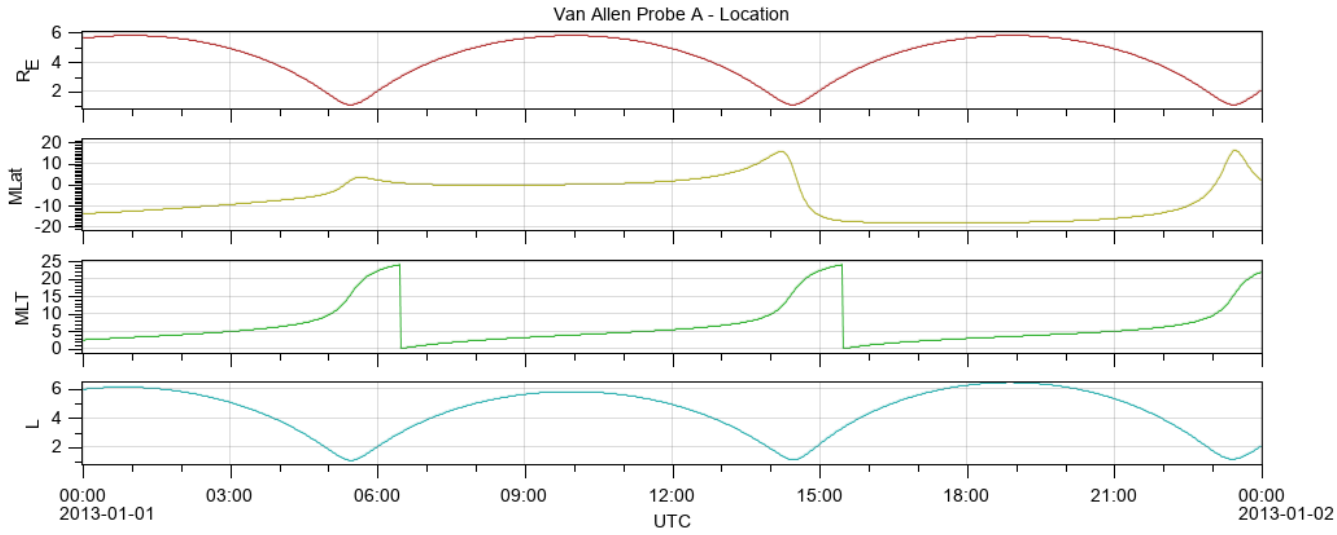


Figure 3.2: An X-Tag stream of Van Allen Probe A location source

3.4.3 X-Scan Example {ver2.3}

New for das2.3, the X-Scan stream below provides an example of transmitting waveform data. As can be seen in the figure below the listing, these are X-Y line plot data and so could have been transmitted as an X-Tag stream in a manner similar to the previous example. However this would result in a considerable waste of network bandwidth as even in binary form, each analog to digital capture is just a four byte binary float, but each one would require an accompanying eight byte double precision time tag. By sending all the time offsets up front, and then just one time tag for each burst of activity, the stream drops to a third of it's original size. To save more space, X-offsets are given by the `xOffsetInterval` attribute instead of spelling out all 6144 of them using the `xOffset` attribute. Also note that all axis attributes have been left in the stream header as axis information cascades (see section 3.4.6).

In the stream listing below each data packet is truncated, as indicated by the ellipses ending each row.

```
[00]000376<stream version="2.3" >
  <properties
     String:title="Waves: 50 kHz Electric Waveforms%{xCacheResInfo}"
     Datum:xTagWidth="40.0 µs"
     String:xLabel="SCET %{RANGE}"
     String:yLabel="E!dy!n Component (V/m)"
     String:yScaleType="linear"
     boolean:xMonotonic="true"
     DatumRange:xCacheRange="2016-08-27T12:51:59.875 to 2016-08-27T12:54:00.124"
 />
</stream>
[01]000192<packet>
  <x type="time24" units="us2000" />
  <xscan name="lfr_lo_e" type="ascii14" yUnits="V m**-1" xUnits="µs"
         nitems="6144" xOffsetMin="0.0" xOffsetInterval="20.0" />
</packet>
:01:2016-08-27T12:52:00.769  1.716286e-02  1.743696e-02  1.825963e-02 ...
:01:2016-08-27T12:52:02.794 -1.323041e-02 -1.359455e-02 -1.377667e-02 ...
:01:2016-08-27T12:52:04.769  2.910113e-03  3.088316e-03  2.643106e-03 ...
:01:2016-08-27T12:52:06.769 -1.293373e-02 -1.275177e-02 -1.311572e-02 ...
:01:2016-08-27T12:52:08.769 -3.715232e-02 -3.779902e-02 -3.807623e-02 ...
:01:2016-08-27T12:52:10.769  2.624610e-03  2.842070e-03  3.166170e-03 ...
:01:2016-08-27T12:52:12.769  1.501577e-01  1.504898e-01  1.505380e-01 ...
:01:2016-08-27T12:52:14.769  8.245772e-04  3.892964e-04  8.245772e-04 ...
```

The these data would typically be plotted as follows.

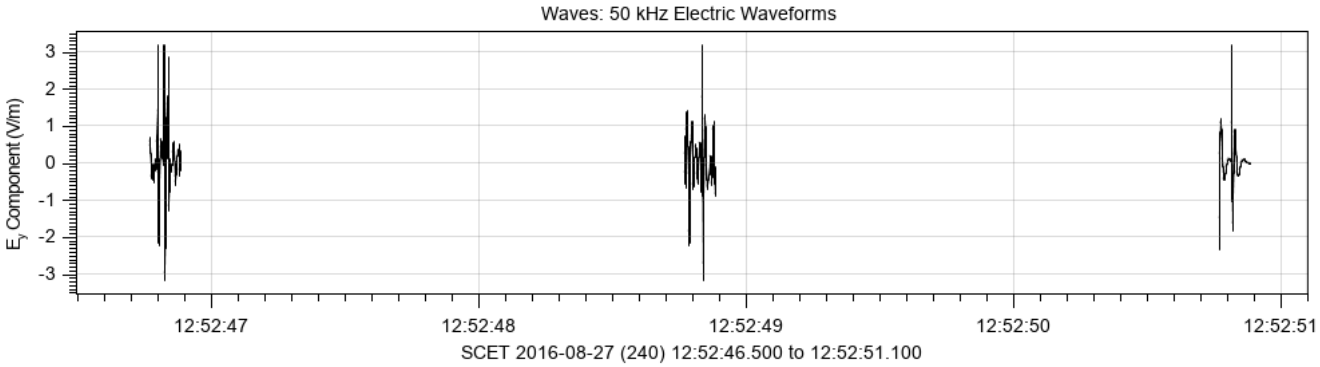


Figure 3.3: X-scan stream from a Juno Waves Low Frequency Receiver source

3.4.4 XY-Tag Example

The following example nicely demonstrates that clients cannot assume <x> arrays contain time values, or that the output is even intended for a Cartesian display. In this instance, plasma densities are provided as a function of Mars Express altitude and the solar zenith angle of the sub-spacecraft point. Even though the das2 server *dataset* query would have required a time range, the output is not in terms of time. Finally, even though the attribute plotHint_00 is not part of this ICD it's okay for the server to include it as it does not conflict with any defined attributes.

```
[00]000370<stream version="2.3">
  <properties
    title="MARSIS Plasma Density by Solar Zenith Angle and Altitude"
    yLabel="Solar Zenith Angle (degrees)"
    xLabel="Altitude (km)" double:xValidMin="0" double:xVaildMax="2000"
    double:yValidMin="180" double:yValidMax="0" zLabel="N!De!N (cm!A-3!N)"
    renderer="polar"       plotHint_00="radius_offset:1000"
/>
</stream>
[01]000167<packet>
  <x type="ascii8" name="altitude" units="km" />
  <y type="ascii8" name="sza" units="degrees" />
  <z type="ascii10" name="dens" units="cm**-3" />
</packet>
:01: 186.49   36.82  1.67e+01
:01: 186.49   36.49  3.13e+01
:01: 186.49   36.16  3.25e+01
:01: 186.49   35.83  3.80e+01
:01: 186.50   35.50  3.55e+01
(many more rows follow...)
```

To help put the stream listing above into visual context, a plot of the full stream output follows.

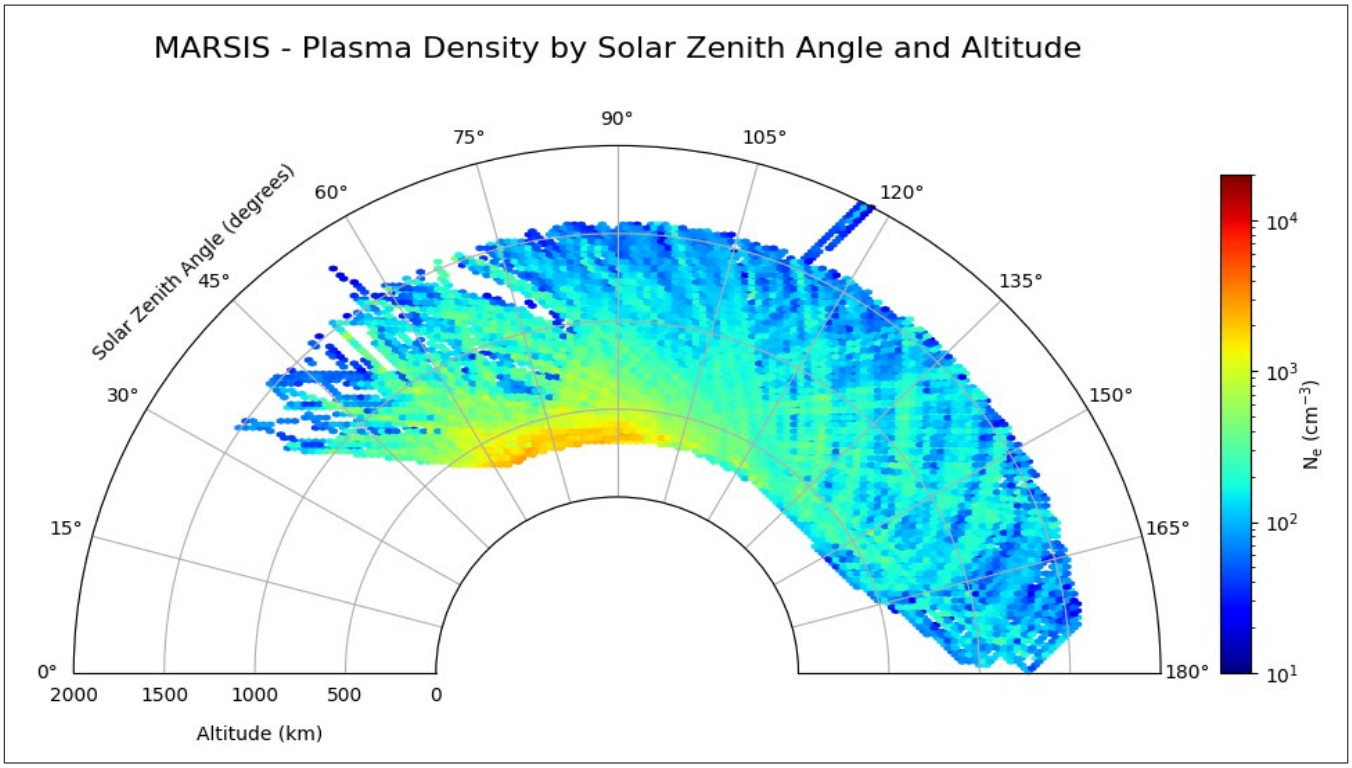


Figure 3.4: Non-time XY-Tag stream from a Mars Express, MARSIS derived data source

3.4.5 XY-Scan Example

Similar to the X-Scan example, this stream transmits a rank-2 array. The X dimension is again time, but the data values are the the Z (i.e. color) direction. The Y offset values are provided *in the Header Packet* instead of in the Data Packets, since there is one Y value in the <yscan> element for each column of data values. Again this is much more efficient then repeating the Y values in each data packet as would be required if using an XY-Tag stream. Since a <y> array is defined, each set of offsets in the <yscan> are added to the <y> reference value in each packet. If the <y> were array missing, the reference point in Y would be taken as 0.0 and the frequency values would be independent of time.

```
[00]000191<stream version="2.3">
  <properties
    title="Raw Waves HFWBR - Perijove 3 Inbound - 512 DFT, 3/4 overlap"
    boolean:xMonotonic="true" yLabel="MHz" Datum:xTagWidth="2.0 s" />
</stream>
[01]000260<packet>
  <x type="time24" units="us2000" name="time" />
  <y type="ascii9" units="MHz" name="frequency" />
  <yscan nitems="4095" type="ascii9" yUnits="kHz" name="HFWBR" zUnits="dn**2/Hz"
         yOffsetMin="-656.25" yOffsetInterval="0.3205128" />
</packet>
:01:2016-12-11T16:20:47.792 4.50e+00 9.07e-01 1.27e+00 8.81e-01 2.50e-01 ...
:01:2016-12-11T16:20:48.792 4.50e+00 6.89e-01 3.20e+00 2.80e+00 3.09e+00 ...
:01:2016-12-11T16:20:49.792 4.50e+00 1.15e+00 1.49e+00 2.14e+00 1.74e+00 ...
:01:2016-12-11T16:20:50.792 4.50e+00 2.48e+00 3.09e-01 1.62e+00 5.91e-01 ...
:01:2016-12-11T16:20:51.792 4.50e+00 3.22e+01 2.03e+01 6.74e+00 4.46e+00 ...
(many more rows follow...)
```

A plot of these "wandering frequency" data can be seen below.

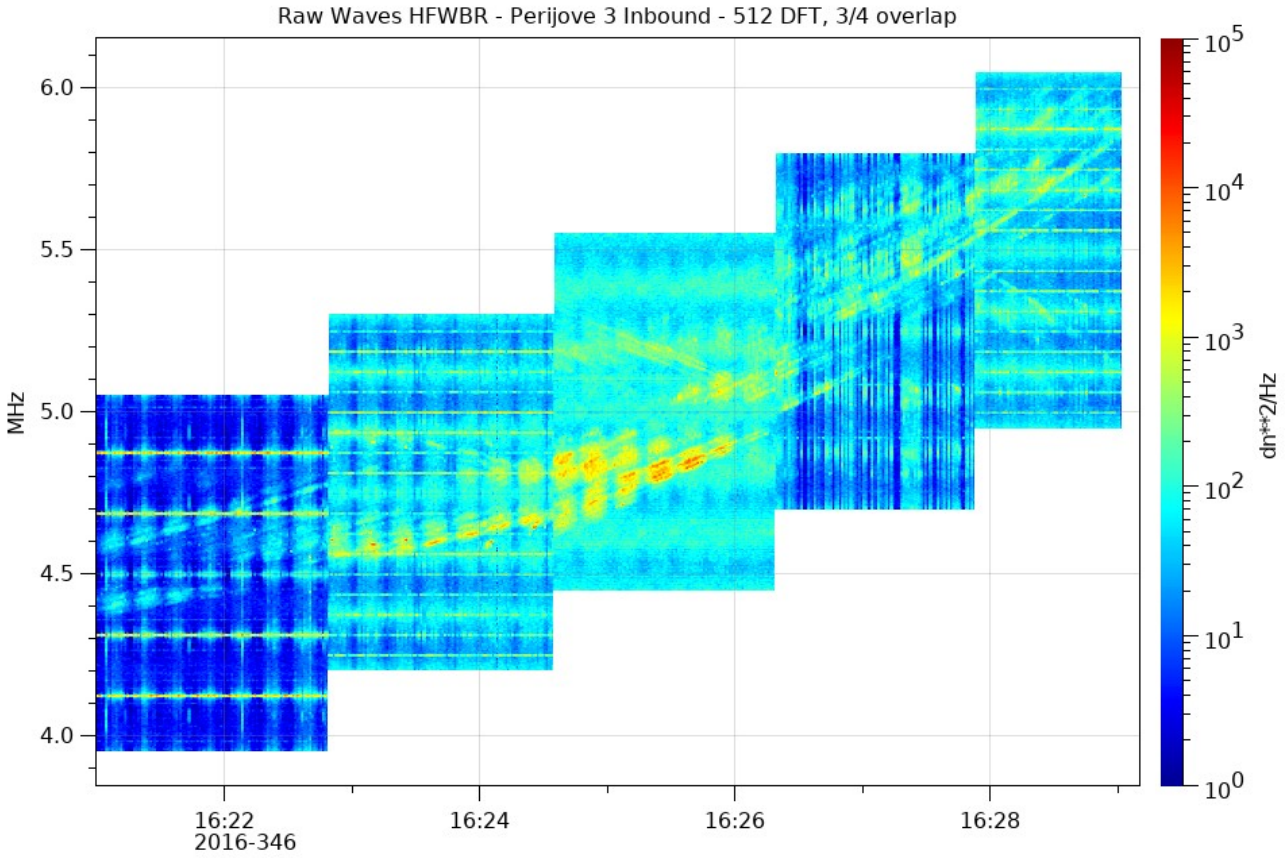


Figure 3.5: XY-Scan stream from a Juno Waves frequency mixing reciever

Client XY-Scan Implementation Note: Versions 2.2 and earlier of the das2 stream format did not define an <xscan> array type. Because of this, some stream sources overloaded the meaning of <yscan> since it was the only table array type available. In those cases, the yTags element provided a set of xOffsets and Y-values (instead of Z-values) occur in the corresponding data packets.

Fortunately, sources that changed the meaning of <yscan> provided a hint in the <stream> element by setting the renderer attribute value to "waveform". For compatibility with older servers, clients are recommended to watch for the value "waveform" in the renderer attribute of the <stream> properties when the version equals "2.2". If this combination of attribute values is detected, the following translations should be enacted while parsing the stream:

```
yscan        -treat-as->  xscan
yTags        -treat-as->  xOffsets
yTagInterval -treat-as->  xOffsetInterval
yTagMin      -treat-as->  xOffsetMin
```

3.4.6 Attribute Cascade

Not only does the top level <stream> attribute contain a <properties> child element, but the data array elements, *<x>*, *<y>*, *<xscan>*, *<z>*, *<yscan>* may have properties as well. When assigning plot labels to array values, programs should look first in the array properties element first for the corresponding property. If the array element does not have an expected property (or has no properties sub-element at all) then programs should consult the top-level <stream> properties element.

Since more than one axis of the same type can be defined in a packet, reading properties in this way this allows common items to be stated once at the top, with overrides for individual arrays. This pseudo listing below demonstrates attribute cascade.

```
[00]000000<stream>
      <properties xLabel="UTC" yScaleType="linear" />
</stream>
[01]000000<packet>
      <x ... />
      <y ... > <properties yLabel="Bx" /> </y>
      <y ... > <properties yLabel="By" /> </y>
      <y ... > <properties yLabel="Bz" /> </y>
</packet>
:01: ...
```

Here each <y> array has it's own Y-axis label. But all of them should be displayed on a linear scale since there is a yScaleType value under the <stream> element.. The <x> array has no properties child element and so it's label can only come from the stream header packet.

3.4.7 Array Element Attribute Tables

The tables below define the XML elements used in das2.3 Header Packets.

Attributes of <x>	
name (required)	Provides a name for the data array, follows the formation rules for C identifiers. If the array values are considered to be independent coordinates (and not the measurements themselves) data soruces are strongly encourage to use simple lowercase standardized names, such as: `altitude energy frequency latitude longitude pitch_angle sza time` The EPN-TAP, CDF or PDS standards may have useful values to consider. {ver2.3}
type (required)	The byte encoding format for the Data Packet values for this array. Four binary types are defined. All four follow the IEEE 754 standard for encoding binary floating point real numbers: • **sun_real8** - 8-byte big-endian value • **sun_real4** - 4-byte big-endian value • **little_endian_real8** - 8-byte little-endian value • **little_endian_real4** - 4-byte little-endian value Two text types are defined: • **time**N • **ascii**N where N is the number of bytes occupied characters in an ASCII time string, for example `time23` or `ascii10`. N is not zero padded. The `time` type is used for time strings and generally should follow the ISO-8601 recommendation. The `ascii` type is used text representations of real and integer numbers. There is no `type` value defined for transmitting character string data in version 2.3 of the ICD.
units (required)	The units of the this array's data values, in general this can be be any reasonable string, but see section 3.4.8 for formation rules for `units` strings. When transmitting exact time points in Coordinated Universal Time (UTC) time frame, the units value should read "`UTC`". For values that are truly dimensionless, use an empty string, i.e. `units=""` as the attribute definition.

Attributes of <x> <properties>	Any attribute from table 3.1 with a name that starts with a lower case "`x`".

Attributes of <y>	
name (required)	Provides a name for the data array, follows the formation rules for C identifiers. If the Y values are considered to be independent coordinates (and not the measurements themselves) servers are *strongly encouraged* to use standardized names. See the name attribute for the <x> element for details.
type (required)	See the description for the `type` attribute in the **<x>** element.

Attributes of <y>	
units (required)	The units for the associated array values in the data packets. See the `units` attribute in the **<x>** element.

Attributes of <y> <properties>	
String:operation (optional)	If the data for associated <y> arrays were derived via some mathematical operation on data from another array, provide the name of the operation here: Currently understood values are: `BIN_AVG, BIN_MAX, BIN_MIN`
String:source (optional)	If the data for associated <z> arrays were derived via some mathematical operation on data from another array, provide the `name` attribute of the original array here. This allows client programs to plot derived statistics together on the same plot for the same upstream source point source.
cascade, optional	Any attribute from table 3.1 with a name that starts with a lower case "`y`".

Attributes of <z>	
name (required)	Provides a name for the data array, follows the formation rules for C identifiers.
type (required)	See the description for the `type` attribute in the **<x>** element.
units (required)	The units for the associated array values in the data packets. See the `units` attribute in the **<x>** element.

Attributes of <z> <properties>	
String:operation (optional)	Associates 2 to N <z> arrays in the same manner as described for this attribute in <y>, <properties> above.
String:source (optional)	Associates 2 to N <z> arrays in the same manner as described for this attribute in <y>, <properties> above.
cascade, optional	Any attribute from table 3.1 with a name that starts with a lower case "`z`".

Attributes of <xscan> {ver2.3}	
name (required)	Provides a name for the data array, follows the formation rules for C identifiers.
nitems (required)	The number of data values per-packet for this array. It is also the number of items in the **xOffsets** attribute as well as the size of the "fast moving index" for this array.
type (required)	See the description for the `type` attribute in the **<x>** element.
xOffsets (optional)	Provides the a rank-1 array of length **nitems** to use as the X-offsets from the <x> reference values on a packet by packet basis. The value portion is a comma separated list of real or integer values, for example: `xOffsets="0.00e+00,2.00e+01,4.00e+01,6.00e+01"`

Attributes of <xscan> {ver2.3}	
	If neither the xOffsets or xOffsetInterval are provided, each offset is the same it's fast moving index value, i.e. 0, 1, 2 ..., `nitems` - 1 .
`xOffsetInterval` (optional)	For X offsets that are a series of points with a fixed interval separating each one, individual xOffsets need not be specified. Instead the interval between points for a single packet of this array's data may be provided using this attribute. For example waveform data typically have linearly spaced time offsets.
`xOffestMin` (optional)	Used to provide the starting point for the `xOffset` sequence that is given by the relation: `X-offset = xOffsetMin + N*xOffsetInterval` for N = 0 through (nitems -1). If not provided take xOffsetMin to be 0.0.
`xUnits` (required)	Used to specify the units of the offset values, *can not be an absolute position unit type such at `t2000`*.
`yUnits` (required)	The units for the associated array values in the data packets. See the `units` attribute in the **<x>** element.

Attributes of <xscan> <properties>	
`String:operation` (optional)	Associates 2-N `<xscan>` arrays in the same manner as described for this attribute in `<y>`, `<properties>` above.
`String:source` (optional)	Associates 2-N `<xscan>` arrays in the same manner as described for this attribute in `<y>`, `<properties>` above.
cascade, optional	Any attribute from table 3.1 with a name that starts with a lower case "`y`".

Attributes of <yscan>	
`name` (required)	Provides a name for the data array, follows the formation rules for C identifiers.
`nitems` (required)	The number of values per-packet for this array. It is also the number of items in the **`xOffsets`** attribute as well as the size of the "fast moving index" for this array.
`type` (required)	See the description for the `type` attribute in the **<x>** element.
`xOffsets` (optional)	Provides the rank-1 array of length **`nitems`** to use as the offsets from the <x> reference value on a packet by packet basis. The value portion is a comma separated list of real or integer values, for example: `xOffsets="0.00e+00,2.00e+01,4.00e+01,6.00e+01"` {ver2.3} If neither `xOffsets` or `xOffsetInterval` are provided, all `xOffsets` are **0.0**
`yTags` (optional)	Provides the rank-1 array of length **`nitems`** to use as the offsets from the <y> reference value on a packet by packet basis. If the <y> reference array is missing, the packet Y reference value is **0.0**. The value portion is a comma separated list of real or integer values, for example: `yTags="0.00e+00,2.00e+01,4.00e+01,6.00e+01"` If **neither** the `yOffsets` nor `yOffsetInterval` (or synonyms) are provided, each offset is the same it's fast moving index value, i.e. **0, 1, 2 ..., `nitems` - 1** .

Attributes of <yscan>	
yOffsets {ver2.3}	A synonym for yTags, included for clarification
xOffsetInterval (optional) {ver2.3}	For X offsets that are a series of points with a fixed value separating each one, individual xOffsets need not be specified. Instead the interval between points for a single packet of this array's data may be provided using this interval. Waveforms and operations on waveforms typically have linearly spaced time offsets.
yTagInterval (optional)	For Y offsets that are a series of points with a fixed interval between each one, individual yOffsets need not be specified. Instead the interval between points for a single packet of this array's data may be provided using this attribute. For example power spectral densities typically have linearly spaced frequency offsets.
yOffsetInterval {ver2.3}	A synonym for yTagInterval, included for clarification
xOffsetMin (optional) {ver2.3}	Used to provide the starting point for the xOffset sequence that is given by the relation: X-offset = **xOffsetMin** + N*xOffsetInterval for N = 0 through (nitems -1). If not provided take xOffsetMin to be 0.0.
yTagMin (optional)	Used to provide the starting point for the xOffset sequence that is given by the relation: Y-offset = y**OffsetMin** + N*y**OffsetInterval** for N = 0 through (nitems -1). If not provided take yOffsetMin to be 0.0.
yOffsetMin {ver2.3}	A synonym for yTagMin, included for clarification
xUnits (may be required) {ver2.3}	Used to specify the units of the xOffset values, *can not be an absolute position unit type such as t2000 or UTC*. Required if X-offsets are given for this array.
yUnits (required)	Used to specify the units of the yOffset values, *can not be an absolution position unit type such as t2000 or UTC*. See the units attribute in the **<x>** element.
zUnits (required)	The units for the associated array values in the data packets, see the units attribute in the **<x>** element.

Attributes of <yscan> <properties>	
String:operation (optional)	Associates <yscan> arrays in the same manner as described for this attribute in <z>, <properties> above.
String:source (optional)	Associates <yscan> arrays in the same manner as described for this attribute in <z>, <properties> above.
cascade, optional	Any attribute from table 3.1 with a name that starts with a lower case "z".

3.4.8 Unit Values

In general, units strings are unspecified, though the use SI units with exponents specified using Fortran format exponent expressions with white space between different physical dimensions is preferred. For example,

```
V**2 m**-2 Hz**-2
```

would be units of electric spectral density. However transmitting any units strings from the source data files is the best way to provide transparency into the original data file units. For example, transmitting '^' characters for exponentiation when reading from CDF files is the best option.

For time values the situation is a bit different, as the units attribute is used to set the epoch time for continuous floating time intervals. Many das2 tools need to convert between different time encodings so the units attribute value is much more significant for those. Known values for this attribute which indicate 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.
- `ns1970` - Nanoseconds 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.
- `UTC` – Used exclusively with text representations of universal coordinate time.

One obvious missing epoch unit type is TT2000 from the CDF standard. If source times are it TT2000 format, convert these to `time`N and transmit as text UTC values.

3.5 Info Packets

In addition to header and data packets, information on stream processing or exceptional conditions may be transmitted using information packets. All Info Packets have the same packet ID, which is not a numeric value. Instead the packet ID consists of the bytes: 0x78, 0x78, i.e. UTF-8 '**xx**'. Info Packet content consists of an XML document. There are two types of info packets readily distinguished by the top level XML element.

3.5.1 Comments

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

Attributes of <comment>	
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.
type (required)	Specifies the type of the comment. At present ther values are understood: ● `"log:info"` – Indicates that the `value` attribute will provide a human readable string for display by the client program during the data load. ● `"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. ● **`"taskprogress"`** – Indicates that the value attribute will provide an integer from 0 to taskSize. Use this comment to set the processing completion level.
value (required)	The value of the value attribute varies based on the type attribute value.

Here's an example stream with embedded comments.

```
[00]000071<stream version="2.3"><properties boolean:xMonotonic="true"/></stream>
[xx]000049<comment type="taskSize" value="100" source=""/>
[xx]000070<comment type="log:info" value="Input:T120101.DAT" source="vgpwrdr"/>
[01]000183<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
```

3.5.2 Exceptions

Exception packets are used to signal to clients that something went wrong. As such they should be the *last* packet in any stream in which they occur. Off nominal conditions that would still allow for stream processing should be communicated via Comment packets instead.

Exception packets contain a single `<exception>` element who's attributes are defined in the table below.

Attributes of <exception>	
type (required)	The type of error. The following error types are understood: ● `"NoDataInInterval"`– The data selection parameters excluded all available data ● `"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. ● "`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. ● ANYSTRING – Any other error type string may but this is merely for documentation or for use by custom clients.
message (required)	The message string for the error.

Here's an example stream that terminates in an exception. In this case a file IO error.

```
 [00]000075<stream version="2.3">
  <properties boolean:xMonotonic="true"/>
</stream>
 [01]000189<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]000103<exception type="IOError"
   message="Directory /opt/project/voyager/pds/VGPW_0101 does not exist" />
```

3.6 Data Packets

Each Data Packet is associated with a preceding Header Packet via the Packet ID. Packet IDs are described in the section 3.2, Packet Prefixes. The Header Packet must be parsed in order to understand how to read it's associated Data Packets.

Each array element in the associated header defined a value encoding using the `type` attribute. As described in section 3.4.7 each `type` is a fixed number of bytes. Each array is either rank-1 and thus has one value per packet, or rank-2 with a fixed size in the "fast moving index". Ragged arrays can not be defined. The consequence of these definitions is that *all data packets* with the *same Packet ID* have the *same length* in bytes, and thus the data packet length is not provided in the packet prefix, it must be determined by reading the corresponding Header Packet. Section 3.2.2 provided the length calculation.

In addition to setting the number of bytes for each value, the `type` attribute also determines the endianness of any binary types. Since each array has it own type attribute, endianness can vary from one value to the next *in the same data packet*. Thus mixed endian streams are possible, though this is rare in practice.

Data packet values are assigned to arrays *in the same order* as the array definition in the corresponding header packet (section 3.4.1). For rank-2 arrays, (i.e. `xscans` and `yscans`) a complete run of values for the fastest moving index occurs consecutively in each packet.

Since a single slice of all the arrays is provided in each data packet, clients should have sufficient information to begin plotting or otherwise processing the data as they stream in, especially for streams that have the **`xMonotonic`** attribute. If clients operate in a slice oriented fashion, there is no limit to the size of the data set that may be processed, and this, after all, is the point of a streaming format.

4 QStream Format

QStreams were introduced after QDataSets were introduced with the das2 application Autoplot. QDataSet is a 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 das2 streams allow the data to be streamed so that processing can be done on the stream as it is received. As with das2 streams, descriptive XML headers are mixed with binary or ASCII data packets.

Differences between QStreams and das2 streams

A few things change with QStreams, namely the stream header identifies which of the data sets is the default data set to be interpreted. Instead of a data set having multiple planes, there are simply multiple data sets and the semantics of QDataSet connect them. Also new, the entire stream must use the same byte order (endianness). Data can be encoded within the header elements, or encoded along with the other data packets. (Das2 streams have the issued that header packets are large when there were many irregularly spaced `yOffsets`.)

4.1 Stream Descriptor

A valid QStream starts with a header. This identifies the endianness of the stream and the default data set. 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 data set 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.

4.2 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 data set of length N where N is the number of [01] packets. MinMax contains two doubles, which are combined to make a rank 2 data set 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
```

4.3 Exceptions

Like das2 streams, 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"/>
```

4.4 Examples

More examples of streams can be found at:

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