
# Das2 PyServer 2.2 - User's Reference

*Revision 2019-08-20*

| Purpose | Who | Date |
|---|---|---|
| Initial Draft | C. Piker | 2015-03-18 |
| Updates to allow for hourly and yearly cache blocks | C. Piker | 2017-05-19 |
| Updates to align with current versions for das2 library and tools | C. Piker | 2017-07-28 |
| Bumped version number, install instruction updates | C. Piker | 2018-03-05 |
| Included Server-Reader interface as this has been removed from ICD | C. Piker | 2019-08-20 |
| | | |
| | | |
| | | |
| | | |

*Recent Revisions*

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

# Table of Contents

1: Overview ..... 4
2: Installation ..... 6
2.1: Prerequisites ..... 6
2.2: Get the Source Code ..... 6
2.3: Build and Install ..... 6
2.4: Configure Apache ..... 7
2.5: Test the Server ..... 8
2.6: Basic Customization ..... 10
3: Data Source Configuration ..... 12
3.1: Reader Program Command Line Interface ..... 12
3.2: Data Source Definition Files (DSDF) ..... 12
3.3: Example DSDF files ..... 16
4: Data Cache Configuration ..... 19
4.1: Enabling a data cache ..... 19
4.2: Cache Layout ..... 19
5: Extra Services ..... 22
5.1: Server-side Plot creation ..... 22
5.2: HAPI Subsystem ..... 22
5.3: ASCII Output Conversion ..... 22
6: Server File Reference ..... 23
6.1: Main Configuration: das2server.conf ..... 23
6.2: Data Set Definition Files: *.dsdf ..... 24
6.3: Server Peers: das2peers.ini ..... 24
6.4: Authentication: passwd, group ..... 24
7: Server Program Reference ..... 26
7.1: Web Service Programs ..... 26
7.2: das2_ascii ..... 27
7.3: das2_srv_arbiter ..... 27
7.4: das2_srv_passwd ..... 27
7.5: das2_srv_todo ..... 27
7.6: das2_bin_avg ..... 28
7.7: das2_bin_avgsec ..... 28
7.8: das2_bin_peakavgsec ..... 29
7.9: das2_cache_rdr ..... 29
7.10: Readers ..... 29
8: Work Queue Reference ..... 30
8.1: Input Work List: das2_todo ..... 30
8.2: In-Process List: das2_working_*proc-id* ..... 32
8.3: Result List: das2_finished ..... 32

| Acronym | Meaning |
|---|---|
| CGI | Common Gateway Interface, A definition of how programs invoked via a Web-Server should interact with that Web-server. See RFC-3875 at http://www.ietf.org/rfc/rfc3875 . |
| DSDF | Data Set Descriptor File, A file defining a Das 2.2 data source for use via das2-pyserver. |
| HTTP | HyperText Transfer Protocol, The basic transport protocol for most of the world's data. |
| NFS | Network File System - A longstanding Linux/Unix file sharing protocol |
| SDDAS | Southwest Data Display and Analysis System |
| | |
| | |

*Table 1: Acronyms used within this document*

| Issue | Affects |
|---|---|
| Windows Installation Instructions have not been written | Section Error: Reference source not found |
| | |
| | |

*Table 2: Known document Issues*

| Document | Purpose | Location |
|---|---|---|
| Das2.2 ICD | Defines the das version 2.2 server-client interface | http://das2.org |
| Heliophysics API | Defines the Heliophysics Application Programming interface client-server interface | |

*Table 3: Related Documents*

# 1: Overview

Das2 servers typically provide data relevant to space plasma and magnetospheric physics research. To retrieve data, an HTTP GET request is posted to a das2 server by a client program and a self-describing stream of data values covering the requested time range, at the requested time resolution, is provided in the response body.

This software, das2-pyserver, provides a caching middleware layer between server-side das2 readers, which stream data at full resolution to their standard output channel, and remote client programs such as Autoplot or SDDAS. The server handles common operations such as network transport, data reduction, authentication and caching, but it relies on specialized reader programs for full resolution data streams.

When using das2-pyserver, the two part das2 client-server system become a three part client-server-reader system. Each part is described below in order of increasing "distance" from the original data files:

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 Definition Files (DSDFs) which tells das2-pyserver how to run the reader.
2. **Das2-pyserver**. This software. It is run via Apache and provides responses for HTTP GET queries. Depending on the information requested and the server configuration, the response may be generated from internal information, or by running a reader.
3. One or more **clients**. These programs are the end destination for the data stream, and are typically plotting programs such as Autoplot or SDDAS, but maybe custom analysis scripts written in Python or IDL.

When handling das2.2 Discovery and Source Definition queries only the DSDF files are read. Readers are not run, this information flow is depicted in Figure 1.1

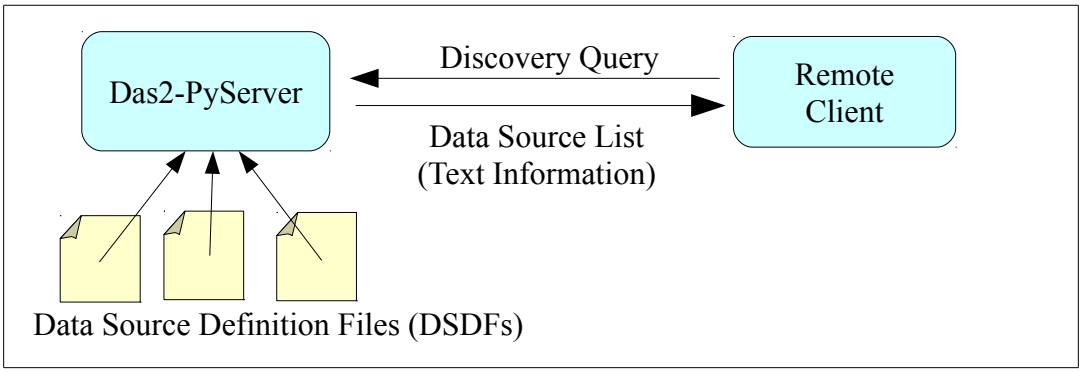


*Figure 1.1 Server handling a das2 discovery query, information source in yellow*

When handling a query for un-cached data values, the server runs an external reader program. Depending on the details of the request, a data reduction program may also be inserted into the output pipeline. This information flow is depicted in Figure 1.2.

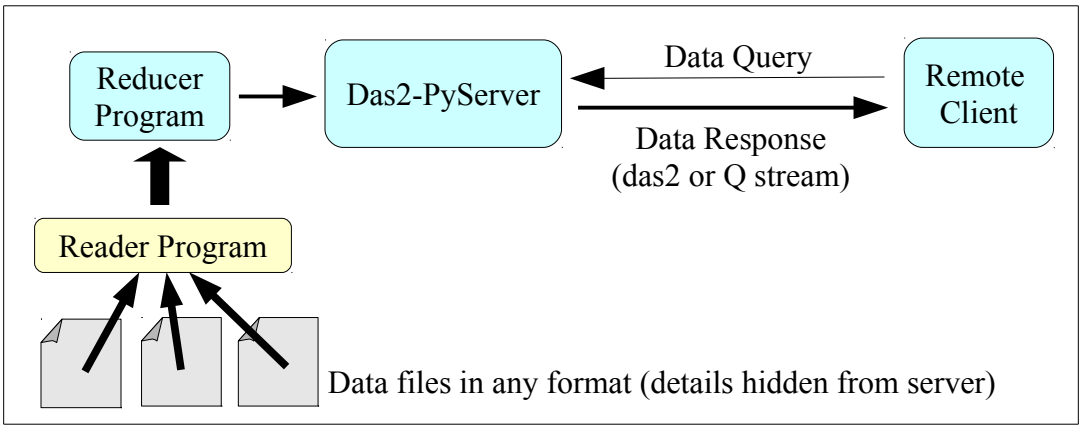


*Figure 1.2 Server handling a das2 data query, information source in yellow*

If data are requested for a time range that has been cached, the server may not run the reader at all. This is useful when long time ranges are requested or when a reader program is particularly inefficient. Whenever the server receives a request for data it checks the data source's DSDF file to see if caching is allowed. If so, das2-pyserver enters a cache request into a Redis job queue and then continues on to supply data is the standard fashion as described above. Subsequent requests for the same time range and similar resolution will read from the cache block files as depicted in Figure 1.3. Depending on the details of the data request this can *significantly* reduce the time needed to satisfy the request.

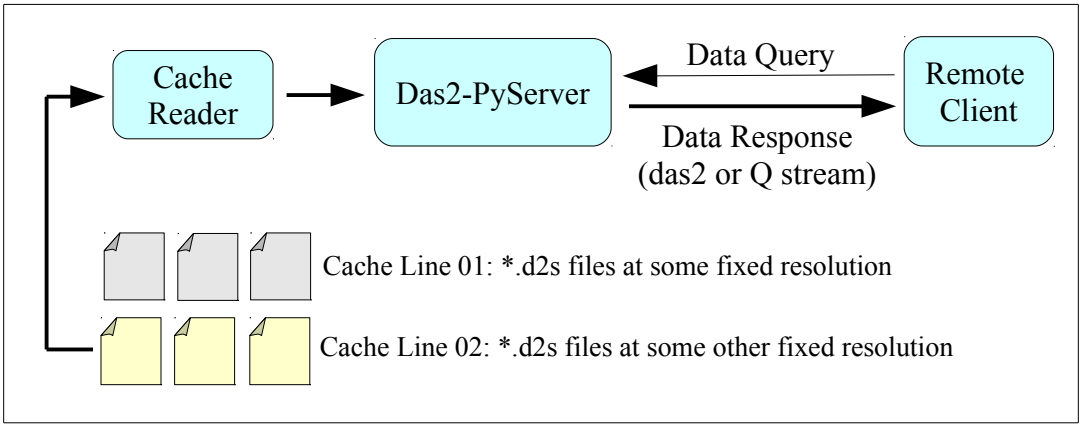


*Figure 1.3 Handling a das2 data query using the pre-reduced cache, information source in yellow*

The first das2 server program was a single-page CGI perl script written in 2002 in support of the University of Iowa's RPWS investigation on board the Cassini spacecraft as part of the das2 data display project. The python version was initially created in support of the EMFISIS instrument on board the twin Van Allen Probes and deployed at the University of New Hampshire. The caching subsystem was written to support the Waves instrument on the Juno mission to Jupiter. As of September 2019, primary data from at least 16 space based instruments and ground observatories as well as location and attitude information are available from various installations of the das2-pyserver software. Das2 client programs and libraries include, but are not limited to, Autoplot[1], SDDAS[2], das2py[3], and das2pro[4].

1 http://autoplot.org, maintained by J.B. Faden, Cottage Systems
2 http://www.sddas.org, maintained by J. Mukherjee, Southwest Research Institute
3 https://das2.org/das2py, maintained by C.W. Piker, The University of Iowa
4 https://github.com/das-developers/das2pro, maintained by C.W. Piker, The University of Iowa

# 2: Installation

Compilation and installation of das2-pyserver software currently requires a Linux environment. That can change if there is ever a demand for using the software on Windows or MacOS.

## 2.1: Prerequisites

Insure that the following programs are installed on your system before beginning the installation procedure.

1. Python >= 2.6, or Python >= 3.4
2. Apache2, any remotely recent version
3. Redis, known to work with version 3.2 or higher
4. redis-py, known to work with version 2.10 or higher
5. libdas2.3, latest version recommended

Since libdas2 provides small binaries needed by das2-pyserver, and since there are no pre-built libdas2.3 packages, installation instructions for both libdas2 and das2-pyserver are included below. In these instructions the '$' character is used at the beginning of a line to indicate commands that you'll need to run in a Bourne compatible shell (bash, ksh, etc.).

Example prerequisite package installation commands are provided below for CentOS 7 ...

```
$ sudo yum install gcc subversion git
$ sudo yum install expat-devel fftw-devel openssl-devel
$ sudo yum install python3 python3-numpy python3-devel  # or python2 equ.
$ sudo yum install redis
$ sudo pip3 install redis                               # if using python3
# --or--
$ sudo yum install python2-redis                        # if using python2
```

... and Debian 9

```
$ sudo apt-get install gcc subversion git
$ sudo apt-get install libexpat-dev libfftw3-dev libssl-dev
$ sudo apt-get install python3-dev python3-distutils python3-numpy  # or py2 eq
$ sudo apt-get install redis-server
$ sudo apt-get install python3-redis                    # or python2 equivalent
```

## 2.2: Get the Source Code

For now, some of the sources are in a University of Iowa SVN server and some are on github.com. All sources will be moved to github.com as time permits.

```
$ svn co https://saturn.physics.uiowa.edu/svn/das2/core/stable/libdas2_3
$ git clone https://github.com/das-developers/das2-pyserver.git
```

## 2.3: Build and Install

Decide where your das2-pyserver code and configuration information will reside. In the example below the directory `/usr/local/das2srv` is selected, but you can choose any location you like. These

environment variables will be used through out the setup, so leaving your terminal window open though the testing stage will save time.

```
$ export PREFIX=/usr/local/das2srv   # Adjust to taste
$ export PYVER=3.6                   # or 2.7, or 3.7 etc.
$ export N_ARCH=/                    # omit per-OS sub-directories
$ export SERVER_ID=solar_orbiter_2   # for ex. ID should not contain white space
```

Test your PYVER setting by making sure the following command brings up a python interpreter:

```
$ python$PYVER
```

The following sequence will build, test, and install libdas2.3 and das2py if you have all prerequisite libraries installed:

```
$ cd libdas2_3
$ make
$ make test
$ make pylib
$ make pylib_test
$ make install
$ make pylib_install
```

Now build and install the python module and example configuration files. Set `--install-lib` and `--prefix` as indicated, unless you want to hand edit `das2server.conf` after installation. There is no need to run build before this step.

```
$ cd ../das2-pyserver
$ python${PYVER} setup.py install --prefix=${PREFIX} \
  --install-lib=${PREFIX}/lib/python${PYVER}
```

Copy over the example configuration file:

```
$ cd $PREFIX/etc
$ cp das2server.conf.example das2server.conf
```

## 2.4: Configure Apache

Apache configurations vary widely by Linux distribution and personal taste. The following procedure is provided as an example and has been tested on CentOS 7.

First determine which directory on your server maps to an Apache HTTPS CGI directory. To do this inspect /etc/httpd/conf/httpd.conf (or similar). The default is `/var/www/cgi-bin`. To provide better URLs for your site add the line:

```
ScriptAlias /das/ "/var/www/cgi-das/"
```

directly under the line:

```
ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
```

inside the `<IfModule alias_module>` section of `httpd.conf`.

Then provide configuration information for your `/var/www/cgi-das` directory inside the `/etc/httpd/conf.d/ssl.conf` file. We're editing the `ssl.conf` instead of `httpd.conf` because das2 clients may transmit passwords.

```
<Directory "/var/www/cgi-das">
  Options ExecCGI FollowSymLinks

  # Make sure Authorization HTTP header is available to Das CGI scripts
  RewriteRule ^ - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]
  RewriteEngine on

  AllowOverride None
  Allow from all
  Order allow,deny
</Directory>
```

By default, authorization headers are not made available to CGI scripts. The re-write rule above allows the Authorization header to be passed down to the das2_srvcgi_main script. This is needed to allow your server to support password protected data sources.

Now symlink the top level CGI scripts into your new CGI directory. Choose the name of the symlink carefully as it will be part of the public URL for your site:

```
$ cd /var/www/cgi-das
$ sudo ln -s $PREFIX/bin/das2_srvcgi_main server
$ sudo ln -s $PREFIX/bin/das2_srvcgi_logrdr log
```

Set the permissions of the log directory so that Apache can write logging information:

```
$ chmod 0777 $PREFIX/log   # Or change the directory ownership
```

Finally, trigger a re-read of the Apache's configuration data:

```
$ sudo systemctl restart httpd.service
$ sudo systemctl status httpd.service
```

## 2.5: Test the Server

Test the server by pointing your web browser at the following two locations:

```
https://localhost/das/server
https://localhost/das/log
```

If this works, try browsing your new server with Autoplot (http://autoplot.org) or SDDAS (http://www.sddas.org).

If you are using Autoplot, copy the URI below into the Autoplot address bar

```
vap+das2server:https://localhost/das/server
```

and then click the green "Go" button. A dialog box similar to the one in Figure 1.4 should appear.

If you are using SDDAS select start gPlot. In the main window click "Sources", then in the sources dialog select Add | Das2. Change the URL in the Das2 Source dialog to:

```
https://localhost/das/source
```

and click the "..." button. You should see a server browse dialog similar to Figure 1.5.

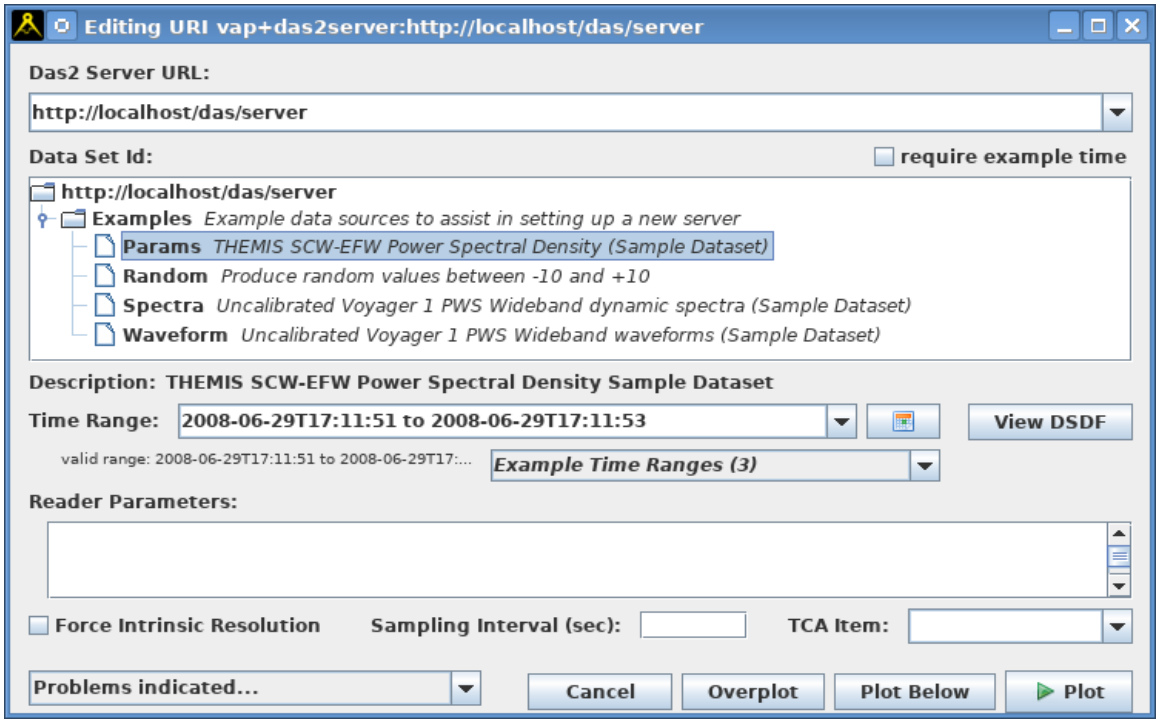


*Figure 1.4 Browsing a new das2-pyserver installation in Autoplot*

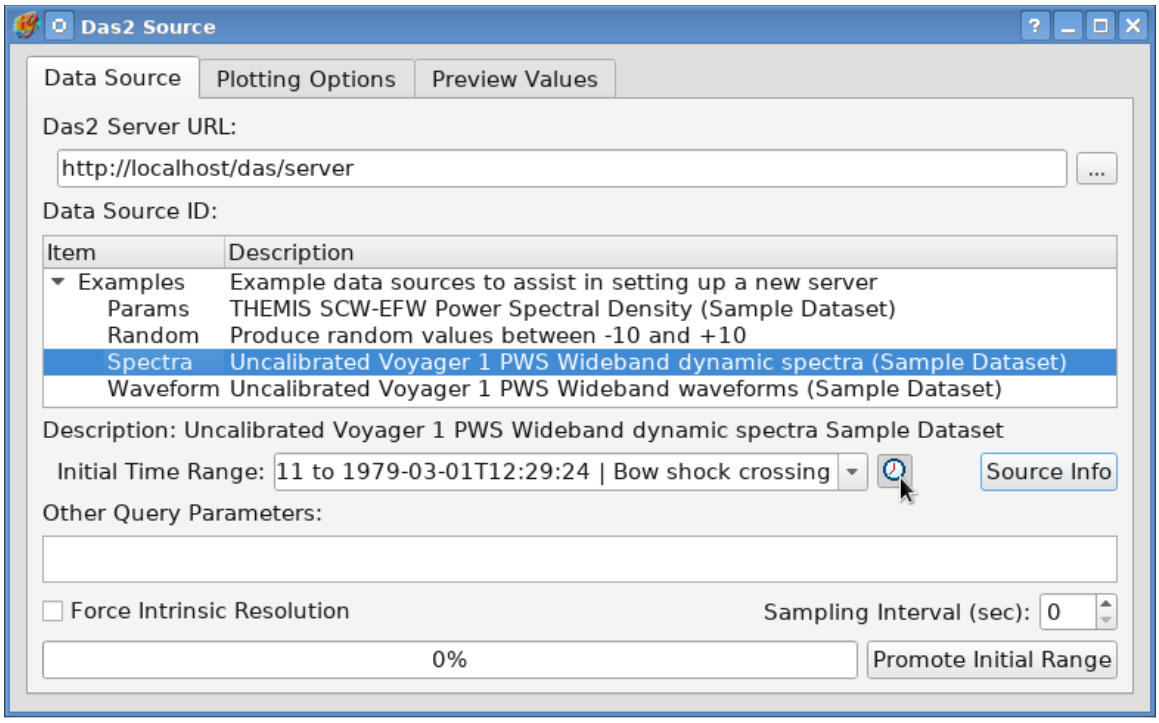


*Figure 1.5 Browsing a new das2-pyserver installation in SDDAS*

The server comes preconfigured with a set of example data sources. Try to load and display a few of these. The Random example should always work no matter the given time range. For the others example times are provided and should be selected in the manner appropriate to your das2 client program.

## 2.6: Basic Customization

### 2.6.1) Configure the Das2 Server Program

1. First setup the peers file

```
$ cd $PREFIX/etc
$ cp das2peers.ini.example das2peers.ini
```

and edit das2peers.ini to taste. Note this is read as a UTF-8 file so go ahead and add non 7-bit ASCII characters if you like.

> *Note: Das2 was developed by English speaking programmers, and thus has not been tested under non English locales. Despite this history, it is the desire of the das2 developers that the software should work with labels and descriptions in any language and thus UTF-8 is taken as the default encoding for all text handling. Please report any problems you encounter with text containing unicode characters at code points above 127. Bugs of this nature will be dealt with promptly.*

2. Next setup the config file

```
$ cd $PREFIX/etc
$ cp das2server.conf.example das2server.conf
```

Point `DSDF_ROOT` to the top of your DSDF file set.

Point `LOG_PATH` somewhere that Apache can write

3. If you have DSDFs with 'server' keywords embedded and you'd like to ignore these in order to test the new server, set `IGNORE_REDIRECT` to true.

4. If you want to enable server-side plot generation un-comment:

```
PNG_MAKER = autoplot_url2png.py
```

in the das2server.conf configuration file and:

```
$ edit $PREFIX/bin/$N_ARCH/autoplot_url2png.py
```

to update any paths that might not make sense on your system. Any program can be used as a server side PNG creator as long as it follows the interface defined in Appendix C.

Currently the configuration file options:

`DSID_ROOT`, `LIB_PATH` and `VIEWLOG2_RELPATH`

don't do anything, you can comment them out or just ignore them.

### 2.6.2) Add a Logo

Copy any file with the pattern logo.* to:

```
$PREFIX/servers/$SERVER_NAME/resources
```

The program will pick it up and even guess the mime-type.

# 3: Data Source Configuration

Das2 PyServer does not read data files directly. Instead it runs programs called readers, which are given query parameters on their command line, gather data from an input source, and emit data streams on their standard output channel. 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:

1. ***program*** **MIN_TIME MAX_TIME [ EXTRA_PARAMETERS... ]**
2. ***program*** **SAMPLING_INTERVAL MIN_TIME MAX_TIME [ EXTRA_PARAMETERS... ]**

Where:

- *program* - any string executable by `/bin/sh`, including pipe (i.e "`|`") characters.
- `SAMPLING_INTERVAL` - For readers marked as requiring time interval values (see **`requiresInterval`** below), this is the expected time interval between data points. The value is an ASCII floating point positive real number greater than 0.0 in units of seconds.
- `MIN_TIME` - The inclusive lower bound UTC time of the query interval. The value should be an ISO-8601 time point string.
- `MAX_TIME` - The exclusive upper bound UTC time of the query interval. The value should be an ISO-8601 time point string.
- `EXTRA_PARAMETERS` - Optional extra parameters sent by the das2 client program. There are no restrictions on these parameters, however pyserver will check them for shell and SQL escapes. 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.

Version 2.2 of das2-pyserver only supports data queries over time, and the command line interface between the server and the reader reflects this limitation. For now, queries over other physical parameters such as latitude and longitude are not supported. A more general command line interface is planned for version 2.3 of the software.

## 3.2: Data Source Definition Files (DSDF)

In order to identify the programs that will supply data streams, das2-pyserver depends on small text files called *Data Source Definition Files* (DSDF). The location of these files is determined by the `DSDF_ROOT` keyword in the `das2server.conf` file. Data source definition files must end in the suffix:

***.dsdf***

or das2-pyserver will ignore them. Though the server is written in python, DSDFs 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. Das2-pyserver ignores any keywords not required for it internal operations. Thus any keyword

may be used, so long as it contains only the characters a-z, A-Z, 0-9 and the underscore, _. A summary of keywords understood by the server and a description of their values is provided in Table 3.1 below. As required by the das2.2 ICD, DSDF information is transmitted to client programs upon request. Extra keywords of use to some client programs are provided in **Table ????**.

| Keyword | Required | Notes |
|---|---|---|
| **description** | **yes** | A human readable description of the data source. |
| **summary** | no | A longer form string describing data processing and other details of the source. The value for this keyword typically extends over many lines and thus makes use of the IDL string concatenation and line continuation syntax, i.e.:<br>`'First line of value, ' + $`<br>`'second line of value.'` |
| **techContact** | **yes** | An email address indicating who to contact if there is a problem with a reader. An example value: 'Some Person <some-person@uiowa.edu>' |
| **sciContact** | no | An email address providing contact information for questions about scientific usefulness or interpretation of the data source. |
| **reader** | **yes** | An arbitrary string that provides the beginning portion of the command line needed to invoke the reader. Additional arguments are supplied by the server as defined in Section 3.1. |
| **requiresInterval** | **maybe** | Set to '1' to indicate that the server should supply the interval between measurements on the reader command line. This triggers use of command line pattern 2 as defined in Section 3.1. The first output time should be MIN_TIME and each successive time should be larger by the value of SAMPLING_INTERVAL. |
| **das2Stream** | **conditional** (yes, if reader outputs a das2 stream) | If set to '1' then the reader produces a das2.2 stream as defined in the Das2.2 Interface Control Document. If the output is a QStream, this keyword should not be present. |
| **qstream** | **conditional** (yes, if reader outputs a Q stream) | If set to '1' the reader produces Q Streams (see section Error: Reference source not found for Q Stream formatting information). There is no need to include this keyword if the reader produces a Das2 Stream. |
| **exampleRange_**XX | **yes** (at least one) | 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. |

| Keyword | Required | Notes |
|---|---|---|
| **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`. |
| **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`. |

| Keyword | Required | Notes |
|---|---|---|
| **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 Error: Reference source not found 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 Error: Reference source not found 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 Error: Reference source not found and Error: Reference source not found 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 Error: Reference source not found for more information on list queries. |
| **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 3.1: DSDF Keywords*

aoeuaoeu

asonteusanoteu

## 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 3.1 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*

# 4: Data Cache Configuration

Typically Das2 Servers exist to feed data to graphical clients. In many situations the data stored on disk are more detailed that an application can display. Take for example a magnetometer than produces one measurement every quarter of a second. Typically a scientist will want to view all data for a particular day, or maybe all data for a multi-day orbit. For our hypothetical instrument, a day's measurements are 345,600 data points. Since an average computer screen is only around 2000 pixels wide sending a whole day's worth of measurements at intrinsic resolution is a waste of network bandwidth and client memory.

As part of the Das2 Client-Server protocol (see the Das 2.2 ICD), client programs may inform the server of the minimum resolution that is expected with each data request. To make better use of resources the Das2 servers use the resolution information to reduce data streams on the server before transmission to the remote client. This makes life easier on the client program, however it does nothing to conserve compute resources on the server. All relevant high resolution files must be read by reader programs for each request, and reduction calculations must be re-run for each client request. To solve this problem, the Das2 PyServer can be directed to cache reduced data streams.

## 4.1: Enabling a data cache

Caching is controlled via keywords in the DSDF file that defines each data source on the Das2 server. The keywords that affect the data cache are given in table 4.1 below.

| DSDF Keyword | Purpose |
|---|---|
| **reader** | Defines the program to run to produce intrinsic resolution data. |
| **reducer** | This is the program which generates the reduced resolution datasets. If not specified a default will be used, based off the stream type. If set to `not_reducible`, then only `intrinsic` resolution caches may be created. |
| **das2Stream** | Let's the server know that the data set reader produces Das2 Streams. Different default reducers, cache writers an cache readers are invoked for different stream types. |
| **qstream** | Let's the server know that the data set reader outputs information in QStreams format. Again this affects the default reducer and cacheReader. |
| **cacheLevel_XX** | Defines the number of resolution levels in the cache and the file storage method used for each one. Each level may specify a resolution in seconds, or `intrinsic` to indicate that reader output should be cached, but not reduced. |
| **cacheReader** | This option is rarely used. It allows an alternate cache reader program to be specified. |

*Table 4.1: DSDF keywords involved in server side caching*

## 4.2: Cache Layout

The following rules are used by Das2 PyServer to store and retrieve files from the disk cache. These are provided in case that you need to manually manipulate the cache, or for writing replacement cache readers.

### 4.2.1) Cache File Path Components

Reduced resolution files are found using the following directory path:

```
CACHE_ROOT / dataset_name / norm_params / res_dir / block_dirs / block_file
```

Table 4.2 below breaks out each component of a cache file path.

| Path Component | Description | Input Source |
|---|---|---|
| CACHE_ROOT | The root of the pre-reduced data file tree | das2server.conf |
| dataset_name | The dataset name on the server. Note that dataset names are hierarchical. Each section of the name becomes a path component. | client query |
| param_string | This is a **normalized** parameter string, thus all spaces and other dangerous components have been transformed into alternate characters. This string does **not** include the data range query paramentes (typically a begin and end time). See part 4.2.2 of this section for normalization rules. | client query |
| res_dir | This directory corresponds to the resolution of the reduced data.<br>• For binned data, the name is the prefix **`bin-`** concatenated with the resolution value (no units).<br>• The server can also cache intrinsic resolution data. In this cane the resolution directory is just named '**`intrinsic`**'. This is a useful optimization when dealing with slow reader programs. | dataset DSDF file |
| block_dirs | These path components are based on the resolution's coverage blocks. For example if the '`daily`' coverage blocks are selected for a resolution in the DSDF file then this would be just be a single directory containing the 4-digit year. See part 4.2.3 of this section for the available schemes. | dataset DSDF file |
| block_file | The filename is produced as follows:<br>*`coverage_resolution.suffix`*<br>where *`coverage`* is scheme dependent, *`resolution`* is the same string found in the **resolution_dir** component above, and suffix depends on the file format, either `d2s` or `qds` for Das2 streams and QStreams respectively. See part 4.2.3 of this section for the available coverage blocks. | dataset DSDF file |

*Table 4.2: Cache File Path Components*

### 4.2.2) Normalized Parameter Strings

todo: document the normalization algorithm

### 4.2.3) Available Coverage Blocks

Currently only time series data may be arranged into coverage periods. This meets must current needs but should be expanded to handle measurements such as radar returns that correlate with locations on a planetary surfaces.

| Coverage Blocks | Directory Pattern | File Name Prefix | Partial Path Example |
|---|---|---|---|
| **persecond** | YYYY/MM/DD/HH/MM | YYYY-MM-DDTHH-MM-SS_ | **2013/10/31/01/05/**2013-10-31T01-05-57_bin-10ms.d2s |
| **perminute** | YYYY/MM/DD/HH | YYYY-MM-DDTHH-MM_ | **2013/10/31/01/**2013-10-31T01-05_bin-1s.d2s |
| **hourly** | YYYY/MM/DD | YYYY-MM-DDTHH_ | **2013/10/31/**2013-10-31T01_bin-1s.d2s |
| **daily** | YYYY/MM | YYYY-MM-DD_ | **2013/10/**2013-10-31_bin-60s.d2s |
| **monthly** | YYYY | YYYY-MM_ | **2013/**2013-10_bin-3600s.d2s |
| **yearly** | (none) | YYYY_ | 2013_bin-86400s.d2s |

*Table 4.3: Times Series Cache Blocks*

Das 2.2 servers are heavily targeted towards handling time series data. It is anticipated that new coverage block storage schemes will be needed to handle future Das 2.3 client queries.

# 5: Extra Services

santeou

## 5.1: Server-side Plot creation

(Initial draft, not complete)

Any program that can take as it's inputs the following command line parameters:

```
server=SERVER_NAME
dataset=DSDF_RELATIVE_PATH
start_time=BEGIN
stop_time=END
image=OUTPUT_FILE_NAME
param=EXTRA_PARAMS
```

and writes a PNG file to the given location can be used by the das2_srvcgi_main script to generate server-side plots which are then transmitted to the client instead of a numeric dataset. As an example, the command line parameters to request an EMFISIS High Frequency Receiver Dynamic Spectrum with noise spikes removed might look like:

```
your_program dataset=rbsp/RBSP-B/HFR_spectra.dsdf  \
      start_time=2013-03-01 \
      end_time=2013-03-02 \
      image=/usr/local/das2srv/tmp/HFR_spectra_34578.png \
      params='KEEP-SPIKES Bu' \
      server=http://emfisis.physics.uiowa.edu/das/das2Server
```

An example wrapper script using the Autoplot program to create plot PNGs is provided in the source code for the server and is build when issuing the commands: "`make aping`" and "`make install_apimg`".

## 5.2: HAPI Subsystem

sanethu

## 5.3: ASCII Output Conversion

# 6: Server File Reference

## 6.1: Main Configuration: das2server.conf

All Das2 PyServer side programs whose name begins with '`das2_srv`' use the das2server.conf file to direct some or all of their actions. The location of this file for a particular Das2 PyServer is compiled into these programs during installation. The location of the file is:

```
$PREFIX/servers/$SERVER_NAME/etc/das2server.conf
```

Here `$PREFIX` and `$SERVER_NAME` are environment variables that were specified during installation and have no default values.

The server also depends various utility programs as well to handle tasks such as converting data streams to ASCII text, up-converting old Das 1 reader output to Das 2.2 format, and reducing data streams on the server before transmission to the client. Utility programs do not consult das2server.conf, but instead must receive all necessary configuration information on the command line.

The only hard-coded path on the server is the configuration file itself. All other locations, including the dataset root directory, the logging directory, the cache location and even the program and library paths are specified in the configuration file. The server is so flexible that the configuration file can even be used to swap in custom code for parts of the main CGI routine! Flexibility like this can break things. If you are having problem's on your server first consult the logs. If that isn't helpful, try using a web-browser to issue a debug query to the server which will look similar to:

```
http://your.host.top/cgi/path/das2Server?server=debug
```

This will cause PyServer to dump it's configuration information to your browser.

**`SITE_IDENTITY`** The name of this Das2 Server. Will be provided to clients that preform a Server ID query on your site.
Default: None

**`DSDF_ROOT`** The path on your local file system to the top of the Data Set Definition file tree.
Default: `$PREFIX/servers/$SERVER_NAME/datasets`

**`CACHE_ROOT`** The path on your local file system to the top of the pre-reduced data set cache.
Default: `$PREFIX/servers/$SERVER_NAME/datasets`

**`RESOURCE_ROOT`** The path on your local file system to the top of the static file resource tree.
Default: `$PREFIX/servers/$SERVER_NAME/resources`

**`USER_PASSWD`** The path on your local file system to the user account password file
Default: `$PREFIX/servers/$SERVER_NAME/etc/passwd`

**`USER_GROUP`** The path on your local file system to the user account group file
Default: `$PREFIX/servers/$SERVER_NAME/etc/group`

**`TEST_FROM`**
Default: None

**MODULE_PATH**

**BIN_PATH**

**LIB_PATH**

**LOG_PATH**

**VIEW_LOG_URL**

**VIEW_LOG_ALLOW**

**D2S_REDUCER**

**D2S_CACHE_RDR**

**D2S_TOASCII**

**QDS_REDUCER**

**QDS_CACHE_RDR**

**QDS_TOASCII**

**WORK_QUEUE_BROKER**

**WORK_QUEUE_CONN**

**SAMPLE_DSDF**

**SAMPLE_START**

**SAMPLE_END**

**INGNORE_REDIRECT**

**PNG_MAKER**

## 6.2: Data Set Definition Files: *.dsdf

## 6.3: Server Peers: das2peers.ini

## 6.4: Authentication: passwd, group

# 7: Server Program Reference

## 7.1: Web Service Programs

### 7.1.1) das2_svrcgi_main

| | |
|---|---|
| **Purpose:** | Main Das2 PyServer CGI program |
| **Usage:** | **das2_srvcgi_main** (All options are set via CGI/1.1 Environment Variables) |
| **Scope:** | Server Specific Program |
| **Config:** | das2server.conf, data set DSDF files |

das2_svrcgi_main is meant to be invoked by a web-server that supports the CGI/1.1 interface standard. Most of the program's activity is directed by two inputs:

1. The CGI environment variables associated with a client request
2. The contents of the das2server.conf file

At present the POST method is not handled by the program, so standard input is not read. Furthermore das2_svrcgi_main does not write to standard error. All error messages out output as HTTP responses on standard output, as is expected of a CGI program. Configuration file details are provided in Section 5, environment variables used by this program are listed below.

| Environment Variable | Summary |
|---|---|
| `HTTP_AUTHORIZATION` (optional) | Used to authenticate client programs if a data source has the keyword `readAccess` in it's DSDF file. (See Das2.2 ICD, Table 5) |
| `HTTP_USER_AGENT` | Used to determine if error messages should be output as Das2 Stream comments or as plain text. |
| `PATH_INFO` | Used to determine which resource file a client program is trying to access.<br>(Das 2.3 protocol will use this for data source location as well) |
| `QUERY_STRING` | Used to determine which action the client is requesting |
| `REMOTE_ADDR` | Used to save log messages by remote IP address and for allowing some IP addresses to skip authentication all together |
| `REQUEST_METHOD` | Current version merely checks to make sure POST is not in use. |
| `SCRIPT_NAME` | Used to determine the URL to the Das2 PyServer itself |
| `SERVER_NAME` | Used to determine the URL to the Das2 PyServer itself |
| `SERVER_PORT` | Used to determine the URL to the Das2 PyServer itself |
| `SERVER_SIGNATURE` | Used to advertise the underlying web-server software on auto-generated web pages. |

#### 7.1.2) das2_srvcgi_logrdr

| | |
|---|---|
| **Purpose:** | Log Reader CGI program |
| **Usage:** | **das2_srvcgi_logrdr** (All options are set via CGI/1.1 Environment Variables) |
| **Scope:** | Server specific program |
| **Config:** | das2server.conf |

## 7.2: das2_ascii

| | |
|---|---|
| **Purpose:** | Server background processing manager |
| **Usage:** | **das2_srv_arbiter** [-h] [--no-daemon] |
| **Scope:** | Server specific program |
| **Config:** | das2server.conf |

## 7.3: das2_srv_arbiter

| | |
|---|---|
| **Purpose:** | Server background processing manager |
| **Usage:** | **das2_srv_arbiter** [-h] [--no-daemon] |
| **Scope:** | Server specific program |
| **Config:** | das2server.conf |

Handles Jobs received from the task queue broker. The format

## 7.4: das2_srv_passwd

| | |
|---|---|
| **Purpose:** | Manage user accounts for Das2 Servers |
| **Usage:** | **das2_srv_passwd** [-h] `[-c] [-b] username [password]` |
| **Scope:** | Server Specific Program |
| **Config:** | das2server.conf |

das2_srv_passwd updates the file indicated by the USER_PASSWD entry in a server's das2server.conf file. At present all password strings are stored using the CRYPT method. Das2 systems are assumed to operate in a somewhat friendly environment. Scientific data are usually not interesting to hackers, and user accounts on Das2 systems mostly exist to prevent premature publishing of data. For these reasons the crypt algorithim is considered sufficient to provide what little security is needed.

## 7.5: das2_srv_todo

| | |
|---|---|
| **Purpose:** | Adds tasks to a Das2 server's task lisk |

| | |
|---|---|
| **Usage:** | **das2_srv_todo** [-h] [--no-daemon] |
| **Scope:** | Server specific program |
| **Config:** | das2server.conf |

Handles Jobs received from the task queue broker. The format

## 7.6: das2_bin_avg

| | |
|---|---|
| **Purpose:** | Reduces the size of Das2 streams by averaging over values in the first <x> plane in a stream. |
| **Usage:** | das2_bin_avg [-b BEGIN] BIN_SIZE |
| **Scope:** | General Program |
| **Config:** | (none) |

## 7.7: das2_bin_avgsec

| | |
|---|---|
| **Purpose:** | Reduces the size of Das2 streams by averaging over time |
| **Usage:** | das2_bin_avgsec [-b BEGIN] BIN_SECONDS |
| **Scope:** | General Program |
| **Config:** | (none) |

das2_bin_avgsec is a classic Unix filter, reading Das 2 Streams on standard input and producing a time-reduced Das 2 stream on standard output. The program averages <y> and <yscan> data values over time, but does not preform rebinning across packet types. Only values with the same packet ID and the same plane name are averaged. Within `<yscan>` planes, only Z-values with the same Y coordinate are combined.

It is assumed that <x> plane values are time points. For this reducer, only the following <x> unit values are allowed:

us2000 - Microseconds since midnight, January 1st 2000

t2000 - Seconds since midnight, January 1st 2000

mj1958 - Days since midnight January 1st 1958

t1970 - Seconds since midnight, January 1st 1970

All time values, regardless of scale, epoch, or representation in the input stream are handled as 8-byte IEEE floating point numbers internally. ASCII times are converted internally to us2000 values.

The BIN_SECONDS parameter provides the number of seconds over which to average <y> and <yscan> plane values. Up to total of 99 <y> and <yscan> planes may exist in each packet type, and up to 99 packet types may exist in the input stream. This is a plane limit, not a limit on the total number of data vectors. <yscan> planes may contain an arbitrary number of vectors. The output stream has the same number of packet types and planes as the input stream, but presumably with many fewer time points.

| Option | Summary |
|---|---|
| -b BEGIN | Instead of starting the 0th bin at the first time value received, specify a starting bin. This useful when creating pre-generated caches of binned data as it keeps the bin boundaries predictable. |

## 7.8: das2_bin_peakavgsec

## 7.9: das2_cache_rdr

| | |
|---|---|
| **Purpose:** | Pre-reduced dataset reader |
| **Usage:** | **das2_srv_cacherdr** [-h] `DATASET_NAME NORM_PARAMS RESOLUTION BEGIN END` |
| **Scope:** | Server specific program |
| **Config:** | das2server.conf, data set DSDF files |

das2_cache_rdr selects pre-generated das2 stream files from a hierarchical data cache of pre-binned data. Full path to cache files is:

```
CACHE_ROOT/dataset_name/norm_params/resolution_dir/block_dirs
```

| Parameter | Summary |
|---|---|
| DATASET_NAME | The name of the dataset, i.e. the relative path to the DSDF file from the root of the DSDF tree. |
| NORM_PARAMS | A normalized string representing the parameters set to the reader. The assumption is that when readers are called with different parameter sets the output data set changes. Each different parameter set is a different set of cache files. The string '`_noparam`' can be used to indicate that no parameters were given to the reader when the cache files were generated.<br>See section , part for a description of reader parameter string normalization. |
| RESOLUTION | An integer providing the resolution requested. The largest bin size that does not exceed this value will be selected as the dataset. The string '`intrinsic`' can be used to select the best resolution available. Also a BINSZ of 0 may be given to select native resolution as well. |
| BEGIN | The starting value of the lookup parameter. |
| END | The ending value of the lookup parameter. |

## 7.10: Readers

# 8: Work Queue Reference

Section 7.3 introduced the program, `das2_svr_arbiter`, which handles background processing requested by other components of a Das2 server. It produces cache files, generates coverage plots, collects usage statistics, etc. By default the program is run as a daemon listening for job requests from a work queue broker. Currently the only work queue broker supported is the Redis server.[5] Thus all remaining information in this section will assume that Redis is in use.

## 8.1: Input Work List: das2_todo

After starting, **`das2_svr_arbiter`** opens a connection to the Redis server listed in the server configuration file, das2sever.conf, and issues a blocking right pop operation on the list variable, `das2_todo`. List values strings defining tasks to be handled and have the following format:

```
req_time|reqester|requester_ex|rmtreq|rmtreq_ex|user|job_cat|job_fields...
```

Task strings always start with the following sub-fields which are delimited by pipe '|' characters.

**0: req_time** - An ISO-8601 time and date string to at least seconds resolution, example: `2015-03-25T10:37:43.334`

**1: requester** - An identifier of the program making the request. For example the URL to the das2server associated with the task may be used: `http://planet.physics.uiowa.edu/das/das2Server`

**2: requester_ex** - Extra information about the server, may be empty.

**3: rmtreq** - Some programs, such as web servers handle request on behalf of remote clients. The IP of the client requesting the operation. Note this may the IP of the server itself if a work item was entered via the **`das2_srv_todo`** program.

**4: rmtreq_ex** - Extra information about a client, such as it's GeoIP location, may be empty.

**5: user** - The user name, if any associated with the work request, may be empty

**6: job_cat** - The job category, may not be empty. Sections 8.1.1 through 8.1.3 below list the current job categories and the job-specific fields

**7-N: job_fields** - Each category of job has it's own list of fields following the job_cat field. See Tables 8.1 through 8.3 for a description of each job category and the required fields.

Empty fields are represented by two adjacent pipes with no intervening characters. To make the protocol expandable, each job category has it's own field set. These are added at index 7 and continue as far as is dictated by the category.

### 8.1.1) Build Cache Task

This task requests a cache build for a specified range of a dataset at one, or all of the resolutions listed in the dataset's DSDF file. This task type is specific to the Das 2.2 system interface. Upcoming Das 2.3 cache tasks will presumably require a different set of fields, or different field values.

Job Category Value: **`TASK_CACHE`**

5 See http://redis.io for more information on Redis, and advanced key-value cache and store server.

| Overall Field Index | Job Field Name | Job Field Description |
|---|---|---|
| **7** | dataset | The name of the dataset that needs to be cached. This is always the relative DSDF filename without the extension. Examples: `juno/wav/survey`, `voyager/1/pws/SpecAnalyzer-4s-Efield` |
| **8** | begin_index | The beginning index to cache, for Das 2.2 this is an ISO-8601 start time. |
| **9** | end_index | The ending index to cache, for Das 2.2 servers this is an ISO-8601 end time. |
| **10** | cache_level | A requested cache level index. This maybe the keyword **`ALL`**, to indicate that all cache levels specified in the dataset's DSDF file are to be produced, or this may be one of the given cache index values. See the **`cacheLevels_XX`** DSDF value in Section 3.2 of the Das 2.2.1 ICD. Leading zeros are ignored. |

*Table 8.1:* ***TASK_CACHE*** *- Task Specific Fields*

The following string provides an example of a cache_2.2 task request:

```
2015-03-25T10:37:43.334|jupiter||128.255.33.104|US,Iowa,Iowa City||TASK_CACHE|
juno/fgm/MagComponentsSCSE||2014-01-01|2014-01-02|ALL
```

### 8.1.2) Record Access Task

Job Category Value: **`TASK_USAGE`**

Access event tasks are meant to record which datasets are being retrieved. The Job specific fields are given in Table 8.2 below.

| Overall Field Index | Job Field Name | Job Field Description |
|---|---|---|
| **7** | dataset | The name of the dataset that needs to be cached. This is always the relative DSDF filename without the extension. Examples: `juno/wav/survey`, `voyager/1/pws/SpecAnalyzer-4s-Efield` |
| **8** | begin_index | The beginning index to cache, for Das 2.2 this is an ISO-8601 start time. |
| **9** | end_index | The ending index to cache, for Das 2.2 servers this is an ISO-8601 end time. |
| **10** | params *(optional)* | A non-indexing parameter string. Some readers take parameters that are not part of the start and stop period of the data cache. |
| **11** | resolution | The floating point resolution datum requested from the client, or empty if no specific resolution was specified. |
| **12** | interval | Some datasets are produced at required intervals only. If such a dataset was accessed this field records the interval, otherwise it is blank. |

*Table 8.2:* ***TASK_USAGE*** *- Task Specific Fields*

The following string provides an example of an access log request for the Voyager spectrum analyzer distogram reader:

```
2015-03-25T10:37:43.334|planet||128.255.33.104|US,Iowa||TASK_USAGE|
voyager/1/pws/SpecAnalyzer-Distogram|2011-07-24|2011-08-31|10 10|340.0 s|
```

### 8.1.3) Update Coverage Plot

Job Category Value: **`TASK_COVERAGE`**

PyServer can provide dataset coverage plots, this work item request an update of those inventory plots.

| Overall Field Index | Job Field Name | Job Field Description |
|---|---|---|
| **7** | dataset | The name of the dataset whose coverage needs to be updated, or the keyword **ALL**, to indicate that coverage for all datasets should be updated. |
| **8** | begin_index | The beginning time for finding data coverage |
| **9** | end_index | The ending time for finding data coverage. |

*Table 8.3:* ***TASK_COVERAGE*** *- Job Specific Fields*

The following string provides an example of a massive coverage plot update.

```
2015-03-25T10:37:43.334|planet||128.255.33.71||cwp|TASK_COVERAGE|||1977-01-01|
2016-01-01
```

## 8.2: In-Process List: das2_working_*proc-id*

Each *instance* of das2_srv_arbiter uses a single list to record jobs in process. The current version of the arbiter is single threaded, so at most one item will appear in a `das2_working_NNNNN` list, it's format is:

```
req_time|server|server_ex|client|client_ex|user|job_cat|job_specific...|begin|
status|progress
```

The working list just adds two fields to the end of which-ever job type is in progress. These are:

**-3: begin** - An ISO-8601 time indicating when the job was started.

**-2: status** - Current job status message. *(optional)*

**-1: progress** - A fraction giving how much of the job is completed, where 1.0 is the entire job. The fraction may be specified as a fraction using the '/' sign to separate two numbers.

Here is an example of a coverage plot task item value.

```
2015-03-25T10:37:43.334|planet||128.255.33.71||cwp|TASK_COVERAGE|||1977-01-01|
2016-01-01|2015-03-25T10:37:44.512|Updating cassini/mag/4-vec|32/114
```

## 8.3: Result List: das2_finished

Finished items end up in a single list for the entire site. This list persists until cleared by a logger. These items are identical to In-Process items with the exception that the progress value is replaced with an end time.

```
req_time|server|server_ex|client|client_ex|user|job_cat|job_specific...|begin|
status|end|return_code
```

**-4: begin** - An ISO-8601 time indicating when the job was started.

**-3: status** - End status message. *(optional)*

**-2: end** - An ISO-8601 time indicating when the job finished.

**-1: return_code** - The finial return code from the handling the work item. To continue the tradition set long ago, 0 indicates success non-zero is a failure code.

```
2015-03-25T10:37:43.334|planet||128.255.33.71||cwp|TASK_COVERAGE|||1977-01-01|
2016-01-01|2015-03-25T10:37:44.512|Generated coverage map for all 114
datasets.|2015-03-29T01:14:55.512|0
```