May, 2016

Dataworks for GNSS:
Manual for Data Repositories

UNAVCO

Dataworks for GNSS: Manual for Data Repositories

by Stuart Wier, Mike Rost, and Fran Boler

UNAVCO

May 1, 2016

Cover illustration:

COCONet Site CN13, San Salvador Baham

Dataworks for GNSS and GSAC are developed and supported by UNAVCO and funded by NSF.

UNAVCO
6350 Nautilus Drive
Boulder, Colorado U.S.A.
www.unavco.org

Table of Contents

1 Introduction to Dataworks for GNSS

Dataworks for GNSS is a software project launched by UNAVCO that helps regional GNSS network operators manage data and metadata for small networks (e.g. tens of stations).

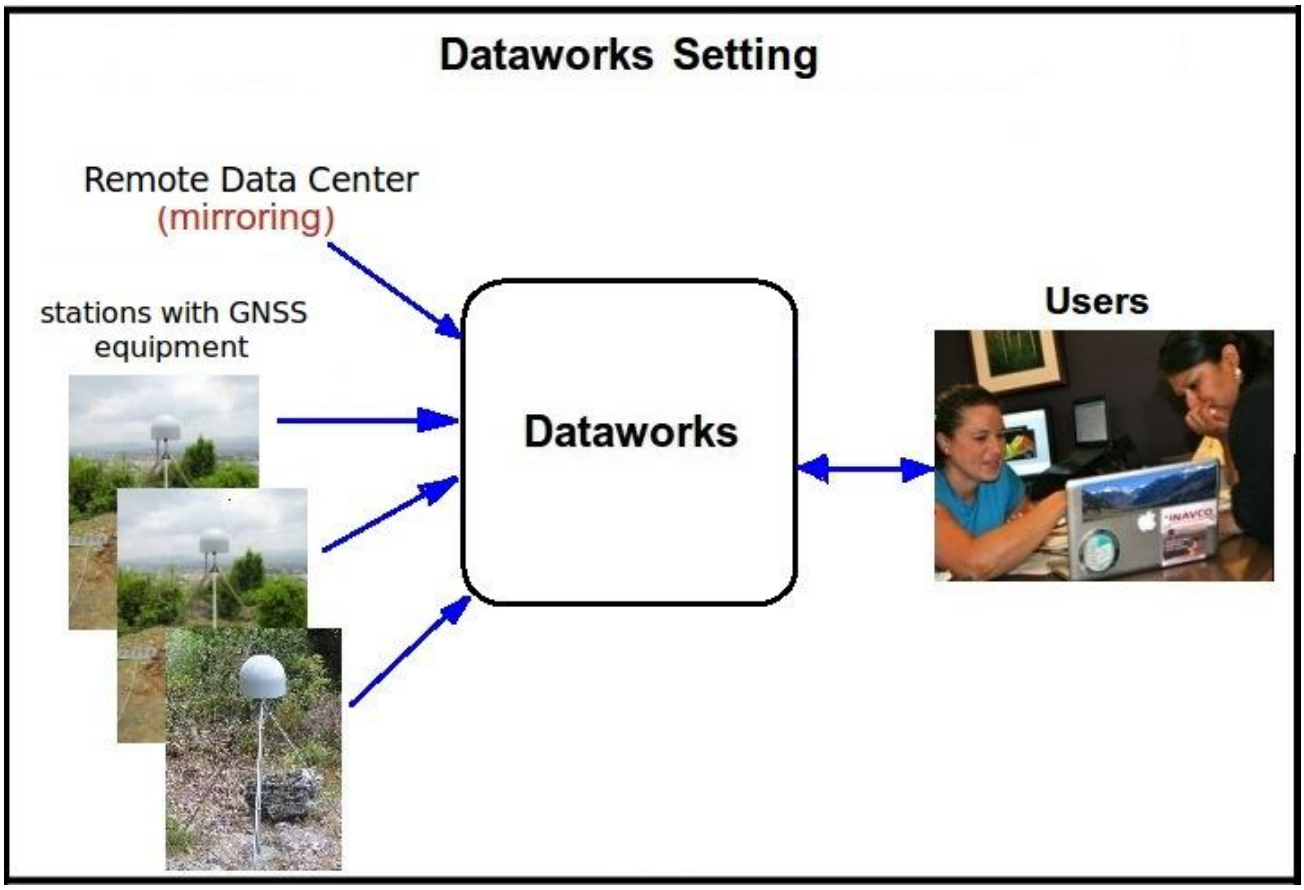
Dataworks for GNSS provides subsystems (e.g. downloading from the receiver and subsequent data management, metadata management using a streamlined database, and data and metadata distribution) as open source software modules or packages. Network operators can select the subsystems that meet their needs for data management functionality. In 2015, UNAVCO provisioned several servers with Dataworks software that were then deployed at data centers in Mexico, Central America and the Caribbean as part of the COCONet and TLALOCNet projects. Following the successful deployment of these systems, the Dataworks software was released publically to interested data centers. The software, copyrighted by UNAVCO, is open source and may be downloaded according to instructions on the UNAVCO web site. The software may be used in accordance with the GNU Lesser General Public License version 2.1 or (at your option) any later version.

The following subsystems of Dataworks are available from the UNAVCO Dataworks web site at http://www.unavco.org/software/data-management/dataworks/dataworks.html, along with documentation:

1. Dataworks for GNSS Database (DGD): schema for MySQL database management system
2. GSAC (Geodesy Seamless Archive Centers): presentation of data and metadata through a web UI for search and access, and as an API
3. Data and Metadata Mirror Modules: mirroring of data and metadata from a GSAC data center and subsequent local data handling
4. Metrics Module: tracking of data ingest and distribution (optional)
5. Data Download Module: receiver downloading and local data handling
6. Data Ingest Module: stores file metadata in the DGD
7. Data Export Module: stores ftp ready files in the distribution file system

A note about examples in this documentation: many examples were drawn from the deployment of the Dataworks software to TLALOCNet and COCONet data centers. Many examples leverage the GSAC and archive contents at UNAVCO. Your utilization can be totally independent of UNAVCO. Any examples referring to COCONet or TLALOCNet are simply for illustration purposes.

The diagram shows an example overall setting for Dataworks including optional “mirroring” of data from a remote data center dunning GSAC.



Dataworks is an interacting set of software modules that require the Linux operating system, software packages, and user logins all to be correctly installed and/or created. There are two options for running Dataworks: a local server that you support, or an Amazon Virtual Machine server running the Dataworks AMI (Amazon Machine Image) that UNAVCO has created; in this case Dataworks will be running "in the cloud".

A running Dataworks instance, whether in the cloud or on a local server, will have a Linux operating system and:

- An Apache web server with the Tomcat Application server
- An FTP service
- A MySQL database
- Java
- Python
- Specific user accounts

The MySQL database comes with the command line client "mysql" that can be used for operating the database and populating information. To assist with populating the Dataworks database, you may wish to install

- MySQL Workbench

To run mirroring scripts you need

- Linux utility "curl"

To use a local server that you support for Dataworks, see the separate document **Dataworks System Services Guide** available at http://www.unavco.org/software/data-management/dataworks/lib/docs/Dataworks_System_Services_Guide.pdf. If you are using your own server, your success with Dataworks will depend on ensuring that each requirement for the server has been set up properly.

Once your server is ready as described in the Systems and Services Guide, you will obtain and install the software by following instructions on the website:

http://www.unavco.org/software/data-management/dataworks/software-repository/software-repository.html

The Dataworks AMI includes everything listed above. All the software is installed and operating when you startup the AMI on your own Amazon virtual machine instance.

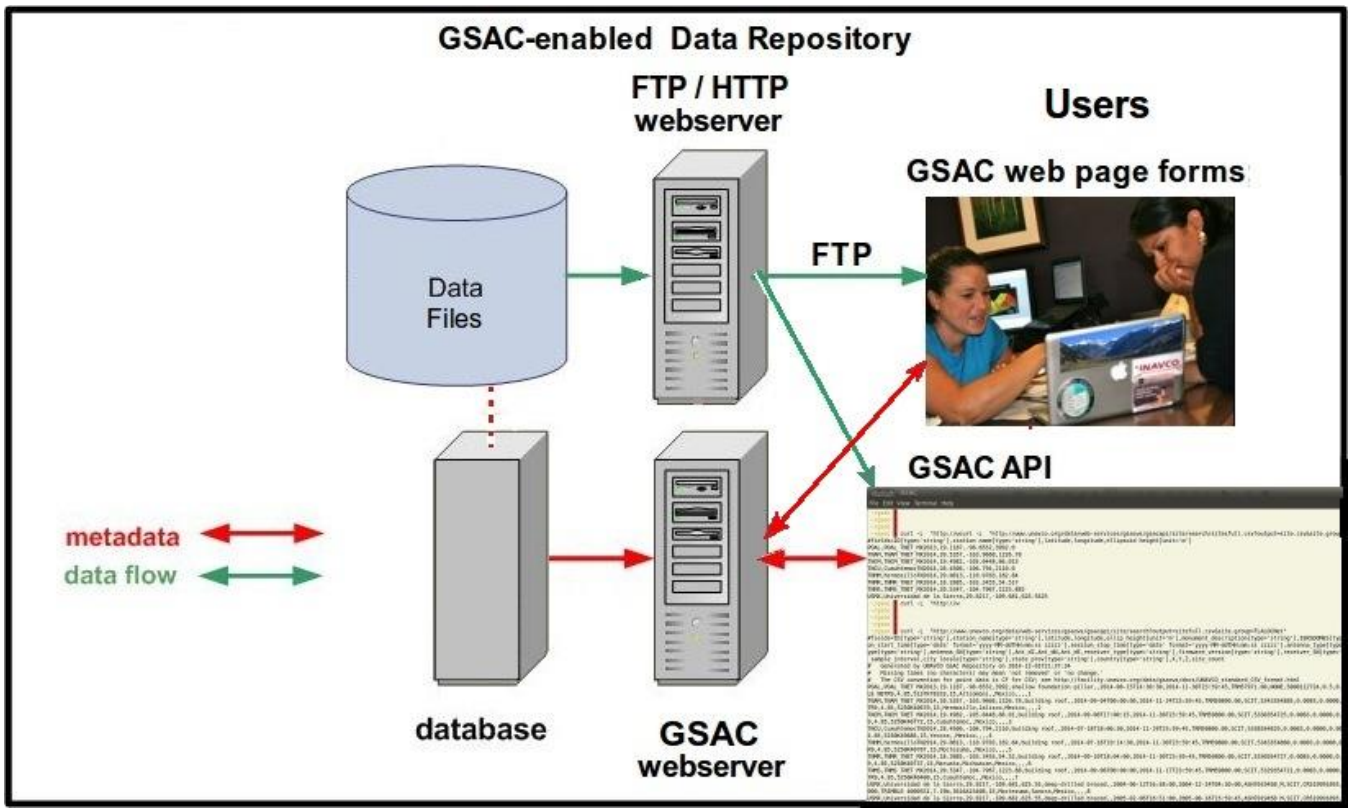
Whether you are using the AMI Dataworks or Dataworks installed on your local system, when you first startup your Dataworks system, the database contains only sample data and you will need to populate the database with station and datafile information and populate any previously acquired data files into your Dataworks data area.

2 GSAC: Discovery and Download Services for Data Repositories

2.1 What is GSAC

GSAC is a software package providing a range of capabilities to connect a data repository to end data users, over the Internet. GSAC allows users to make requests to find information about GPS stations (sites), instruments, and data files. Users may download GPS receiver data files using file URLs from GSAC. **Familiarity with GSAC documentation** http://www.unavco.org/software/data-management/gsac/gsac.html **is highly recommended. Exploring the working UNAVCO GSAC is useful, too.**

This diagram shows a data repository with GSAC, data files, the Dataworks database, and an FTP server. Users make requests via a web user interface or though a command line shell or terminal commands (an API, or Application Programming Interface request).



Users query GSAC for information (metadata) about sites (stations), instruments, and instrument data files. Users receive information about sites (stations), instruments, and downloading instrument data files. Note that no data files flow through GSAC. GSAC tells users how to get data files with FTP URLs.

For example, UNAVCO has a GSAC (at http://www.unavco.org/software/data-management/gsac/gsac.html). In the context of the COCONet Dataworks deployment, GSAC at UNAVCO is used by COCONet centers to find information about COCONet stations and data files, and load the local COCONet data repository database, mirroring data originating at the UNAVCO Archive. And each COCONet data center has its own GSAC to allow remote users to find the information about COCONet stations and data files held at that

COCONet data center.

The GSAC package of services, like the database, is built with the concepts of ***stations*** (sites or monuments), at ***fixed locations*** (latitude and longitude), with *instruments* (GNSS receivers), making instrument ***data files***.

A data center with GSAC has a ***database with metadata*** about the stations, instruments, and data files; data files for public download on an FTP server; a web server; GSAC software; and a server (hardware, which could be in the cloud). GSAC receives requests, reads from the database, and sends results to remote end users. Though GSAC can operate with many databases if they hold the necessary metadata, in this documentation, use with the Dataworks for GNSS database schema (DGD) (Section 3) is assumed, and supported, while use with other databases is not covered or supported.

GSAC only reads from the database. Inserting data in the database, and database maintenance, is handled outside of GSAC. GSAC has no write permissions in the database. Part of operating GSAC is database maintenance and keeping the database up to date, as described in Section 3.

Note that GSAC does not "know about" the local data file system. GSAC does not read data files. GSAC does not know data file formats. GSAC does not know about data file naming conventions. GSAC does not alter, understand, or modify any kind of data files, including RINEX or other types of GPS data files. GSAC does not check data file quality, correct files, or otherwise manage files or manage a data archive or a database.

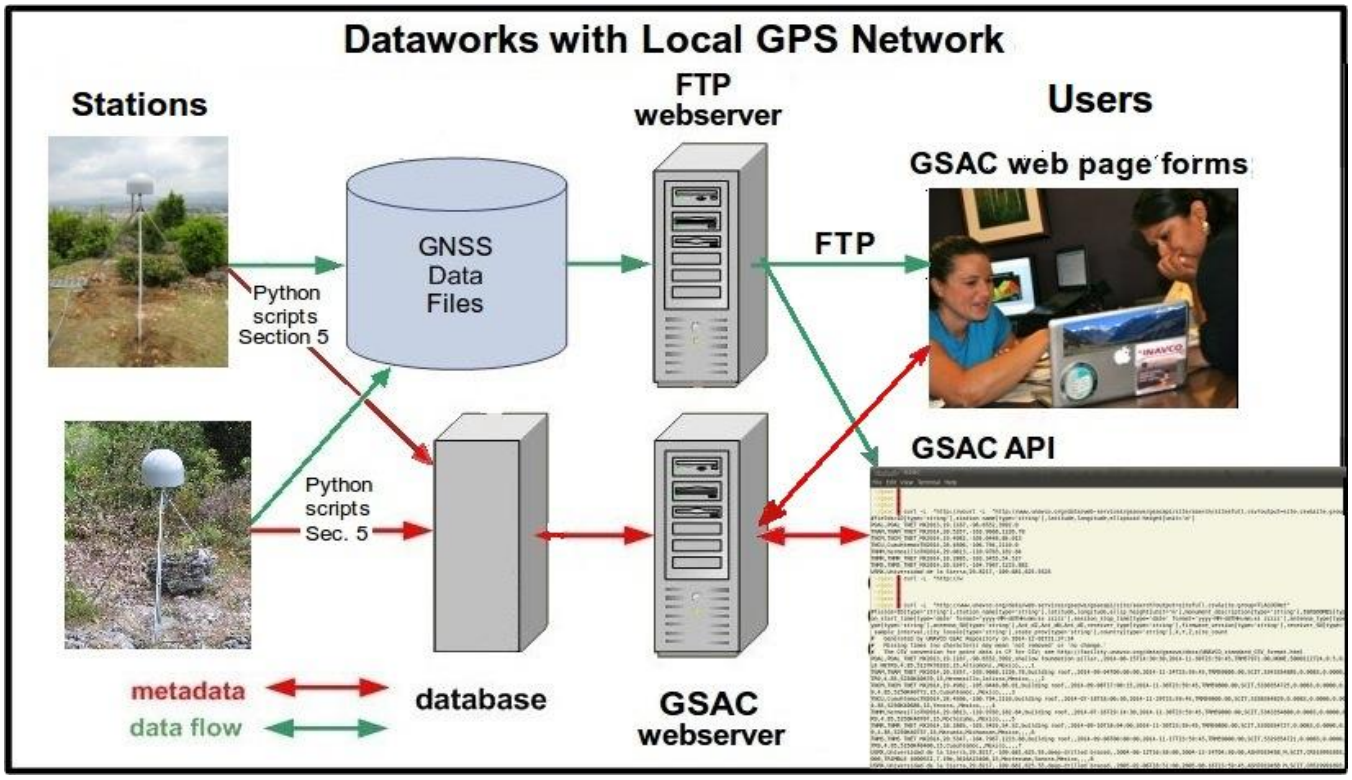
GSAC is designed to be a general-purpose access tool for GNSS stations, instruments, and data files. Though GSAC can work with many kinds of geodetic and other instrument networks and many kinds of data files, this does not apply for Dataworks for GNSS, which is a set of software tailored for GNSS.

GSAC does not "know about", or directly use the local FTP file server. GSAC does not handle file downloads. GSAC does provide URLs, stored in the DGD, to users, so they can download files. But the URLs may be at any or several data centers, so far as GSAC is concerned.

For complete details about GSAC concepts and how to use GSAC, see the UNAVCO GSAC web site http://www.unavco.org/software/data-management/gsac/gsac.html, and the GSAC User Guide: http://www.unavco.org/software/data-management/gsac/lib/docs/UNAVCO_GSAC_User_Guide.pdf

2.2 Local Station Support with GSAC in Dataworks for GNSS

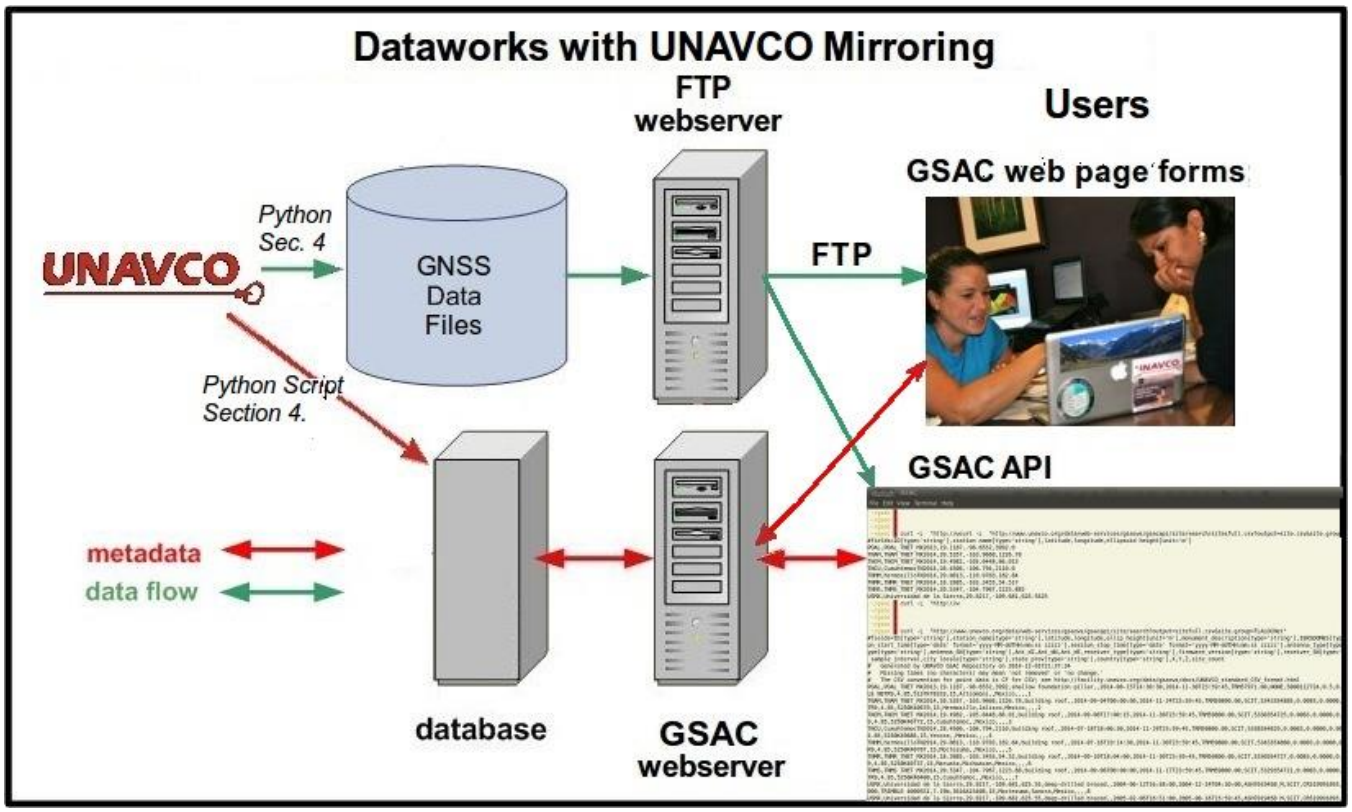
In the context of Dataworks for GNSS, GSAC is part of the larger structure shown in the diagram below. Dataworks for GNSS installations can download and manage data files from local GNSS stations and these data files are accessible through GSAC. For handling local data, a set of Python script retrieves station data files and updates the database. The database must be maintained by the data center with correct and complete information about the local stations and their instruments. The Python code is provided by UNAVCO as part of Dataworks for GNSS, and is described in Section 5 below.



2.3 GSAC in Dataworks for GNSS at Mirror Archives

For mirror archives, Dataworks for GNSS also uses a remote GSAC (often the UNAVCO GSAC) via its API to obtain station and instrument information, and to download the associated GNSS data files from the remote GSAC. Metadata about stations and instruments is queried from the remote GSAC with a Python script accessing the remote GSAC API, to populate and update the DGD (the red line from "UNAVCO" in the diagram; alternatively, another non-UNAVCO GSAC could be the source for mirroring). The Python script is provided by UNAVCO as part of Dataworks for mirrors, and is described in Section 4 below.

Also as also part of "mirroring" the remote GSAC data, GNSS data files from station instruments are copied from the remote GSAC/data center with another Python script, to populate and update the DGD (the green line from UNAVCO in the diagram). This second Python script is provided by UNAVCO as part of Dataworks for GNSS for mirrors, and is described in Section 4 below.



The processes indicated by the red and green lines from UNAVCO, in the diagram above, act to accomplish "mirroring" UNAVCO's archive.

2.4 Installing and Building GSAC

The following sections are a quick overview of GSAC installation and will not usually be part of routine use of Dataworks for GNSS because most of the installation is handled already.

Note: Dataworks server setup should include creating an account for the user with the control of the Dataworks software, that is, the user having responsibility to ensure Dataworks software is up to date and tested. This user is designated in this documentation as having user name "developer1".

When using an Amazon image or after obtaining and installing Dataworks according to instructions in a separate document, you will find the instructions to build GSAC in two README files:

`Dataworks-SW/dataworks-gsac/src/org/gsac/README (the "GSAC README part 1"), and`
`Dataworks-SW/dataworks-gsac/src/org/dataworks/gsac/README ("README part 2").`

Note the system requirements in the "GSAC README part 1" file.

Complete details to install GSAC are in those two README files. The complete instructions are not duplicated here.

2.5 Configuring GSAC Web Site Appearance (Optional)

To create a customized web style within GSAC for your particular Dataworks repository, you change a few files which control the header and footer of web pages created by GSAC. The header and footer are html files:

`gsac-code/src/org/dataworks/gsac/resources/header.html`

`gsac-code/src/org/dataworks/gsac/resources/footer.html`

These files use standard HTML. Edit these files to specify text and images in your GSAC web pages' headers and footers. All GSAC web pages have the same header and footer. Sample working code, including use of logo image files, was included in these files supplied by UNAVCO. Note that logos and other image files are kept in

`gsac-code/src/org/dataworks/gsac/htdocs.`

That directory also has a file index.html which has the contents of the GSAC home page.

After you revise any of these html files, rebuild GSAC again i.e. do command 'ant' again in

`gsac-code/src/org/dataworks/gsac/`

and if it succeeds, then copy the new GSAC war file to the Tomcat area as described above.

Always test your GSAC after any rebuild of GSAC. First click on the blue choice "Search Sites" on the top menu line. This brings up the web page to do site searches. On that page, click on the grey "Search Sites" button. This should show a table of all the sites in your database. If your database has not been populated, no sites are listed. Then click on the blue choice "Search Files" on the top menu line. This brings up the web page to do searches for data files in your repository. On that page click on the grey "Search Files" button. This should show a table of hundreds of data files in your repository. If your database has not been populated, no files are listed. Next, use the "Base URL" for your GSAC, in your browser, such as http://tlalocnet.udg.mx/tlalocnetgsac/, the part before "gsacapi/" in a full GSAC URL. This will show your GSAC's home page. The header, contents, and footer should be what you want for your GSAC web site. These simple tests should indicate that GSAC is working, even if your database has not been populated.

3 The Dataworks for GNSS Database (DGD)

A central component of Dataworks for GNSS is a relational database with information about stations (sites), instruments, and GNSS data files from instruments. The software components of Dataworks use the database for supporting nearly all the Dataworks functionality. Dataworks for GNSS uses a MySQL database engine with a schema or instance named "Dataworks." Making any changes to the Dataworks schema is not supported and will likely cause failure of the Dataworks software.

When you obtain the DGD schema from the Dataworks software repository, and startup a MySQL instance with the Dataworks schema, the fundamental tables will be empty; several of the lookup tables will be prepopulated with commonly used information such as receiver and antenna models.

The DGD is designed around four central concepts and three fundamental tables plus numerous supporting tables. The central concepts and fundamental tables are:

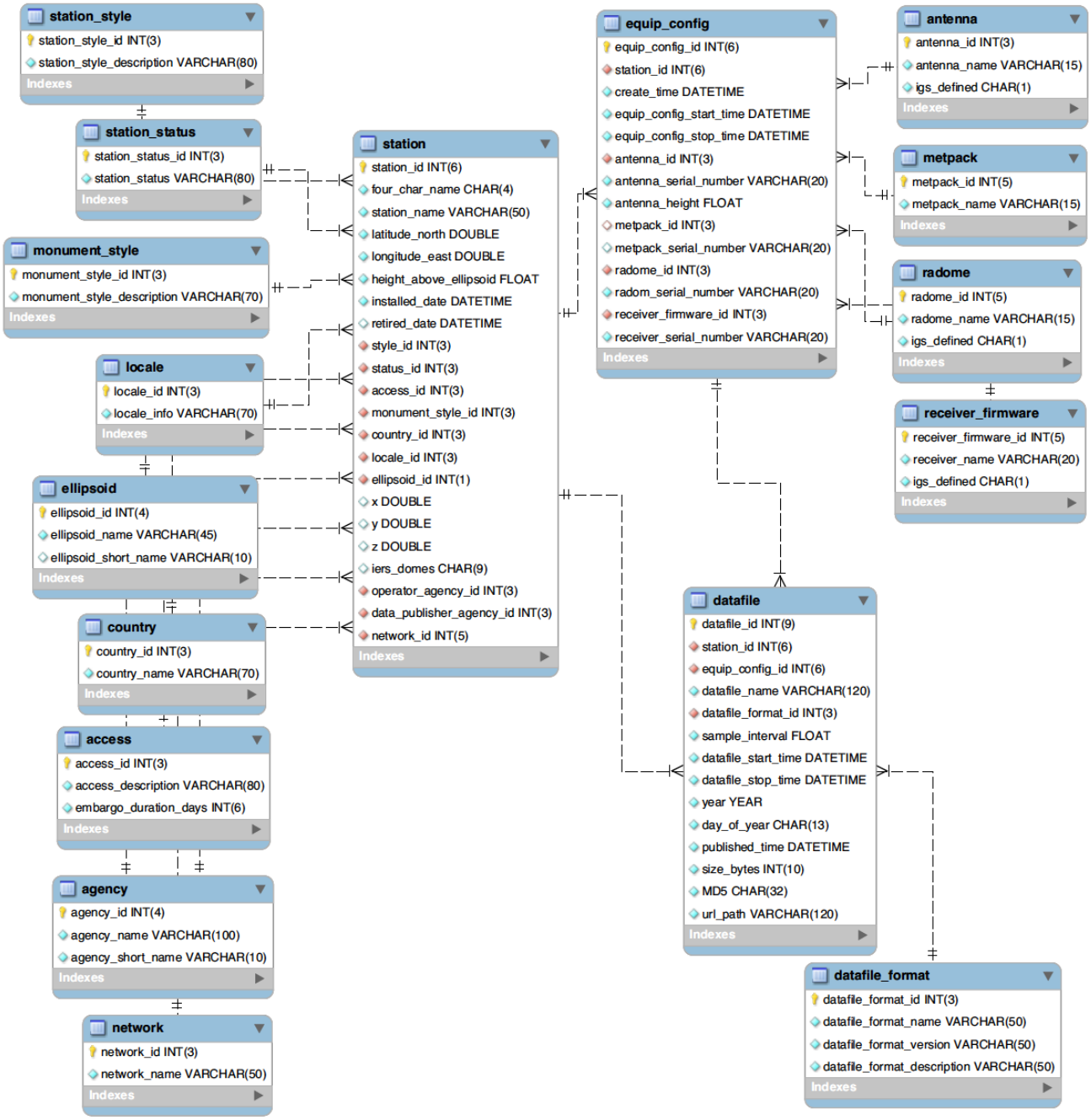
***metadata**:* metadata is information about stations, instruments (equipment sessions), and observational data files. Metadata describes configurations, specifications, and operations. Metadata is *not* a data file. You are a person. Your name is metadata about you, but your name is not you. A picture of a pipe is metadata about that pipe. A picture of a pipe is not a pipe.

station: a known position, where GNSS equipment (at minimum a receiver and an antenna) are installed for collecting data. The DGD **station** table is a fundamental table.

equipment session: An "equipment session" is associated with a particular station. An equipment session is the metadata about the instrumentation at one station during a time span when the instrumentation collects data and has no modification. The DGD **equip_config** table is a fundamental table.

data files: a data file holds observational data values from an instrument at a station, such as a RINEX obs file from GPS receiver. Data files have GPS receiver observational data, such as epochs and pseudoranges. Data files are files on disk somewhere. Data files have a file name, a disk path location, and a size in bytes. Data files are not metadata. Note that streamed data are not currently handled by Dataworks. The DGD **datafile** table is a fundamental table.

A diagram of the full database tables and their fields and data types are shown in the figure.



DGD schema diagram.

The DGD holds all the metadata. Station metadata is in table 'station.' The table 'equip_config' contains information about instrumentation at a station, during "equipment sessions" at each station. A row in the equip_config table details one equipment session for one station. Each 'datafile' table row has metadata about one data file from one station during one equipment session which is described in one equip_config table row. There typically are many data files for each equipment session.

Many of the DGD tables use a “primary key’; in this case, each row in the table has a column whose name is '*table_name*_id' which can be used to easily reference one particular row in that table. Several tables' rows may have values that are also foreign keys to other tables, like station_id or equip_config_id.

Each field has its own data type. Data types help insure that correct or allowed values are entered in the database. For example, you cannot insert "Arecibo Observatory" in the station table field 'four_char_name,' or "4.64 North" into latitude_north, or "Colombia" into country_id, a number. Any field name with "_id" is a key in one or more tables, and all "_id" values are integers.

Note that some fields may have value NULL, meaning unspecified. Most fields must have a non-null value.

3.1 Fundamental Tables

Table 'station' details the basic and infrequently modified information about a GNSS observing site.

The commands like "describe station;" are MySQL command line tool commands. MySQL commands end with ";".

```
describe station;
```

Field	Type	Null	Key	Default	Extra
station_id	int(6) unsigned	NO	PRI	NULL	auto_increment
four_char_name	char(4)	NO		NULL	
station_name	varchar(50)	NO		NULL	
latitude_north	double	NO		NULL	
longitude_east	double	NO		NULL	
height_above_ellipsoid	float	NO		NULL	
installed_date	datetime	NO		NULL	
retired_date	datetime	YES		NULL	
style_id	int(3) unsigned	NO	MUL	NULL	
status_id	int(3) unsigned	NO	MUL	NULL	
access_id	int(3) unsigned	NO	MUL	NULL	
monument_style_id	int(3) unsigned	NO	MUL	NULL	
country_id	int(3) unsigned	NO	MUL	NULL	
locale_id	int(3) unsigned	NO	MUL	NULL	
ellipsoid_id	int(1) unsigned	NO	MUL	NULL	
iers_domes	char(9)	YES		NULL	
operator_agency_id	int(3) unsigned	YES	MUL	NULL	
data_publisher_agency_id	int(3) unsigned	YES	MUL	NULL	
network_id	int(5) unsigned	NO	MUL	NULL	
station_image_URL	varchar(100)	YES		NULL	
time_series_URL	varchar(100)	YES		NULL	

To create a new row (station) in the station table, note the field names ending in _id. These refer to a row in some other table; an example is network_id, which must refer to (have a value set to the network_id value of) a row already entered in the table 'network.' To create a new row in the table 'station,' first you must have a value for every '_id' value, for example, you must have a value in the 'country' table with a country_id value to enter in the station table. You cannot first enter some new value for country_id in the station table, and then add the new value to table country. The "_id" values are created automatically when

you do a correct insertion in a table. There are ten such "_id' values in table station. The value station_id is automatically created when you succeed in inserting a new row, with all required values to table station. The web location of images of a site photograph and of a site position times series plot, if available, may be stored in fields station_image_URL and time_series_URL. See Appendix A for more about MySQL commands.

Table equip_config has the metadata about instrument equipment session at the stations.

```
describe equip_config;
+------------------------+-----------------+------+-----+---------+----------------+
| Field                  | Type            | Null | Key | Default | Extra          |
+------------------------+-----------------+------+-----+---------+----------------+
| equip_config_id        | int(6) unsigned | NO   | PRI | NULL    | auto_increment |
| station_id             | int(6) unsigned | NO   | MUL | NULL    |                |
| create_time            | datetime        | NO   |     | NULL    |                |
| equip_config_start_time | datetime       | NO   |     | NULL    |                |
| equip_config_stop_time | datetime        | YES  |     | NULL    |                |
| antenna_id             | int(3) unsigned | NO   | MUL | NULL    |                |
| antenna_serial_number  | varchar(20)     | NO   |     | NULL    |                |
| antenna_height         | float           | NO   |     | NULL    |                |
| metpack_id             | int(3) unsigned | YES  | MUL | NULL    |                |
| metpack_serial_number  | varchar(20)     | YES  |     | NULL    |                |
| radome_id              | int(3) unsigned | NO   | MUL | NULL    |                |
| radome_serial_number   | varchar(20)     | NO   |     | NULL    |                |
| receiver_firmware_id   | int(3) unsigned | NO   | MUL | NULL    |                |
| receiver_serial_number | varchar(20)     | NO   |     | NULL    |                |
| satellite_system       | varchar(20)     | YES  |     | NULL    |                |
| sample_interval        | float           | YES  |     | NULL    |                |
+------------------------+-----------------+------+-----+---------+----------------+
```

sample values:

```
| equip_config_id | station_id | create_time         | equip_config_start_time |
|             147 |         37 | 2014-02-12 21:51:45 | 2014-02-12 21:51:45     |

 equip_config_stop_time | antenna_id | antenna_serial_number | antenna_height | metpack_id |
 2014-04-06 23:59:45    |          9 | 5151354359            |         0.0083 |          1 |

 metpack_serial_number | radome_id | radome_serial_number | receiver_firmware_id |
 C5010015              |         3 |                      |                   10 |

 receiver_serial_number | satellite_system |
 5145K79595             | GPS              |
```

Field 'station_id' indicates the station which has or had this equipment session. Field create_time is when the row was entered in the database. Fields equip_config_start_time and equip_config_stop_time are the time limits of an equipment session.

Table datafile has the metadata about GNSS data files in this data repository.

```
describe datafile;
+-------------------------+----------------+------+-----+---------+----------------+
| Field                   | Type           | Null | Key | Default | Extra          |
+-------------------------+----------------+------+-----+---------+----------------+
+---------------------------+----------------+------+-----+---------+----------------+
| datafile_id               | int(9) unsigned | NO   | PRI | NULL    | auto_increment |
| station_id                | int(6) unsigned | NO   | MUL | NULL    |                |
| equip_config_id           | int(6) unsigned | YES  | MUL | NULL    |                |
| datafile_name             | varchar(120)    | NO   |     | NULL    |                |
| unique_info_id            | int(9)          | YES  |     | NULL    |                |
| original_datafile_name    | varchar(100)    | YES  |     | NULL    |                |
| datafile_type_id          | int(3) unsigned | NO   | MUL | NULL    |                |
| sample_interval           | float           | YES  |     | NULL    |                |
| datafile_start_time       | datetime        | NO   |     | NULL    |                |
| datafile_stop_time        | datetime        | NO   |     | NULL    |                |
| year                      | year(4)         | NO   |     | NULL    |                |
| day_of_year               | int(3)          | NO   |     | NULL    |                |
| published_time            | datetime        | NO   |     | NULL    |                |
| size_bytes                | int(10)         | YES  |     | NULL    |                |
| MD5                       | char(32)        | NO   |     | NULL    |                |
| URL_path                  | varchar(120)    | NO   |     | NULL    |                |
| data_originator_url_domain | varchar(50)    | YES  |     | NULL    |                |
+---------------------------+----------------+------+-----+---------+----------------+
```

sample values:

```
| datafile_id | station_id | equip_config_id | datafile_name   | original_datafile_name |
|        1923 |         37 |             147 | cn250590.14d.Z | cn250590.14d.Z         |

 datafile_type_id | sample_interval | datafile_start_time | datafile_stop_time  | year |
                2 |               0 | 2014-02-28 00:00:00 | 2014-02-28 23:59:45 | 2014 |

 day_of_year | published_time      | size_bytes | MD5                              |
          59 | 2014-03-01 00:00:00 |     675113 | bf963478361cf3fad32daf88c2c5e04c |

 URL_path                                                  | data_originator_url_domain |
 ftp://dataworks1/pub/rinex/obs/2014/059/cn250590.14d.Z | www.unavco.org             |
```

Each datafile row has one corresponding station (indicated with value station_id), and one corresponding equip_config (indicated with value equip_config_id). These values must exist in the database before you insert a new row in table datafile. The field URL_path is supplied by GSAC to a remote user; it tells where the datafile is available.

Note that GNSS data files have two kinds of compression, as in the file cn250590.14d.Z above. The "d" in ".14d" indicates Hatanaka compression, a special technique for GNSS files. The ".Z" is another compression, with the Compress Linux or Unix shell compression program.

3.2 Lookup Tables

Lookup tables hold information to be associated with station, equip_config, or datafile rows. The DGD has prepopulated rows for these tables. You may need to add rows, which you are free to do with the SQL insert statement, or other means.

```
desc datafile_type;
```

```
+--------------------------+----------------+------+-----+---------+----------------+
| Field                    | Type           | Null | Key | Default | Extra          |
+--------------------------+----------------+------+-----+---------+----------------+
| datafile_type_id         | int(3) unsigned | NO   | PRI | NULL    | auto_increment |
| datafile_type_name       | varchar(50)    | NO   |     | NULL    |                |
| datafile_type_version    | varchar(50)    | NO   |     | NULL    |                |
| datafile_type_description | varchar(50)   | NO   |     | NULL    |                |
+--------------------------+----------------+------+-----+---------+----------------+
```

The mysql command 'select *' shows all the fields in all the rows in a table:

```
mysql> select * from datafile_type;
+-----------------+-----------------------------+-+-------------------------------------------------+
| datafile_type_id | datafile_type_name | datafile_type_version| datafile_type_description
|
+-----------------+-----------------------------+---------------------
|            1  | instrument data file         |                      | Any type or format of native, raw, or
binary file  |
|               2 | RINEX observation file       |                      | a RINEX 'o' obs file; may be
compressed           |
|               3 | RINEX GPS navigation file    |                      | a RINEX 'n' nav file; may be
compressed           |
|               4 | RINEX Galileo navigation file |                     | a RINEX 'e' nav file; may be
compressed           |
|               5 | RINEX GLONASS navigation file |                     | a RINEX 'g' nav file; may be
compressed           |
|               6 | RINEX meteorology file       |                      | a RINEX 'm' met file; may be
compressed           |
|               7 | RINEX QZSS navigation file   |                      | a RINEX 'j' nav file; may be
compressed           |
|               8 | RINEX Beidou navigation file |                      | a RINEX 'c' nav file; may be
compressed           |
+-----------------+-----------------------------+---------------------
```

Insert a new row for a new data file type if you have other types. Note that data type "RINEX observation file" may include plain (ASCII) Rinex files, or such files compressed with Hatanaka compression, or compressed with the Compress Linux utility, or compressed with both compression schemes. You can tell which kind of file you have with the file extension or final character in the extension.

```
 describe access;
+---------------------+----------------+------+-----+---------+----------------+
| Field               | Type           | Null | Key | Default | Extra          |
+---------------------+----------------+------+-----+---------+----------------+
| access_id           | int(3) unsigned | NO   | PRI | NULL    | auto_increment |
| access_description  | varchar(80)    | NO   |     | NULL    |                |
| embargo_duration_days | int(6)       | NO   |     | 0       |                |
+---------------------+----------------+------+-----+---------+----------------+
select * from access;
+-----------+-------------------------------------------------------------------------------+
|  access_id | access_description                                                            |
+-----------+-------------------------------------------------------------------------------+
|          1 | no public access allowed                                                     |
|          2 | public access allowed for station metadata, instrument metadata, or data files
```

```
|          3 | public access allowed for station and instrument metadata only               |
+------------+------------------------------------------------------------------------------+
```

Access is a static table; you do not add or alter values in this table. Access table values can be used to hide information from remote users in GSAC. In Dataworks for GNSS the field 'embargo_duration_days' can be used to hide information from remote users in GSAC for a period of time. These functions require special coding in the local GSAC code.

```
 describe agency;
+-------------------+-----------------+------+-----+---------+----------------+
| Field             | Type            | Null | Key | Default | Extra          |
+-------------------+-----------------+------+-----+---------+----------------+
| agency_id         | int(4) unsigned | NO   | PRI | NULL    | auto_increment |
| agency_name       | varchar(100)    | NO   |     | NULL    |                |
| agency_short_name | varchar(10)     | NO   |     | NULL    |                |
+-------------------+-----------------+------+-----+---------+----------------+
select * from agency;
+-----------+-------------------------------------------------+-------------------+
| agency_id | agency_name                                     | agency_short_name |
+-----------+-------------------------------------------------+-------------------+
|         0 | NOT AVAILABLE                                   | NA                |
|         2 | Institut Geographique National                  |                   |
|         5 | University of Puerto Rico, Mayaguez             |                   |
...
+-----------+-------------------------------------------------+-------------------+
```

Add a new row to the agency table when needed, for a new station with a new agency.

```
describe country;
+--------------+-----------------+------+-----+---------+----------------+
| Field        | Type            | Null | Key | Default | Extra          |
+--------------+-----------------+------+-----+---------+----------------+
| country_id   | int(3) unsigned | NO   | PRI | NULL    | auto_increment |
| country_name | varchar(70)     | NO   |     | NULL    |                |
+--------------+-----------------+------+-----+---------+----------------+
select * from country;
+------------+--------------------------+
| country_id | country_name             |
+------------+--------------------------+
|          2 | France                   |
|          3 | British Virgin Islands   |
|          4 | Panama                   |
|          5 | Montserrat               |
...
```

Add a new row to this table when needed, for a new station with a new country. This is done automatically by Dataworks mirroring software,

```
describe ellipsoid;
+---------------------+-----------------+------+-----+---------+----------------+
| Field               | Type            | Null | Key | Default | Extra          |
+---------------------+-----------------+------+-----+---------+----------------+
| ellipsoid_id        | int(4) unsigned | NO   | PRI | NULL    | auto_increment |
| ellipsoid_name      | varchar(45)     | NO   |     | NULL    |                |
| ellipsoid_short_name| varchar(10)     | YES  |     | NULL    |                |
+---------------------+-----------------+------+-----+---------+----------------+
```

```
select * from ellipsoid;
+-------------+---------------+--------------------+
| ellipsoid_id | ellipsoid_name | ellipsoid_short_name |
+-------------+---------------+--------------------+
|           1 | WGS 84         | WGS 84             |
|           2 | GRS 80         | GRS 80             |
|           3 | PZ-90          | PZ-90              |
+-------------+---------------+--------------------+
```

```
describe locale;
+-------------+-----------------+------+-----+---------+----------------+
| Field       | Type            | Null | Key | Default | Extra          |
+-------------+-----------------+------+-----+---------+----------------+
| locale_id   | int(3) unsigned | NO   | PRI | NULL    | auto_increment |
| locale_info | varchar(70)     | NO   |     | NULL    |                |
+-------------+-----------------+------+-----+---------+----------------+
mysql> select * from locale;
+-----------+-------------------------------------+
| locale_id | locale_info                         |
+-----------+-------------------------------------+
|         2 | LES ABYMES                          |
|         3 | Anegada                             |
|         4 | Sherman                             |
|         5 | Panama City                         |
...
```

Locales are place names. The locale_info can be, for example, a place, city, or town name. Add a new row to this table when needed, for a new station. This is done automatically if needed by Dataworks mirroring software.

```
describe monument_style;
+--------------------------+-----------------+------+-----+---------+----------------+
| Field                    | Type            | Null | Key | Default | Extra          |
+--------------------------+-----------------+------+-----+---------+----------------+
| monument_style_id        | int(3) unsigned | NO   | PRI | NULL    | auto_increment |
| monument_style_description | varchar(70)   | NO   |     | NULL    |                |
+--------------------------+-----------------+------+-----+---------+----------------+
select * from monument_style;
+------------------+-----------------------------+
| monument_style_id | monument_style_description |
+------------------+-----------------------------+
|                2 | building roof               |
|                3 | deep foundation pillar      |
|                4 | deep-drilled braced         |
...
```

Add a new row to this table when needed, for a new station. This is done automatically if needed by Dataworks mirroring software.

```
describe station_status;
+------------------+-----------------+------+-----+---------+----------------+
| Field            | Type            | Null | Key | Default | Extra          |
```

```
+-------------------+-----------------+------+-----+---------+----------------+
| station_status_id | int(3) unsigned | NO   | PRI | NULL    | auto_increment |
| station_status    | varchar(80)     | NO   |     | NULL    |                |
+-------------------+-----------------+------+-----+---------+----------------+
select * from station_status;
+-------------------+-----------------------+
| station_status_id | station_status        |
+-------------------+-----------------------+
|                 1 | Active                |
|                 2 | Inactive/intermittent |
|                 3 | Retired               |
|                 4 | Pending               |
+-------------------+-----------------------+
```

This is a static table; you should not need to add to it or alter values in it.

```
describe station_style;
+--------------------------+-----------------+------+-----+---------+----------------+
| Field                    | Type            | Null | Key | Default | Extra          |
+--------------------------+-----------------+------+-----+---------+----------------+
| station_style_id         | int(3) unsigned | NO   | PRI | NULL    | auto_increment |
| station_style_description | varchar(80)    | NO   |     | NULL    |                |
+--------------------------+-----------------+------+-----+---------+----------------+
select * from station_style;
+------------------+--------------------------+
| station_style_id | station_style_description |
+------------------+--------------------------+
|                1 | GPS/GNSS Continuous      |
|                2 | GPS/GNSS Campaign        |
|                3 | GPS/GNSS Mobile          |
+------------------+--------------------------+
```

This is a static table; you should not need to add to it or alter values in it.

```
describe antenna;
+-------------+-----------------+------+-----+---------+----------------+
| Field       | Type            | Null | Key | Default | Extra          |
+-------------+-----------------+------+-----+---------+----------------+
| antenna_id  | int(3) unsigned | NO   | PRI | NULL    | auto_increment |
| antenna_name | varchar(15)    | NO   |     | NULL    |                |
| igs_defined | char(1)         | NO   |     | N       |                |
+-------------+-----------------+------+-----+---------+----------------+
select * from antenna;
+------------+--------------+-------------+
| antenna_id | antenna_name | igs_defined |
+------------+--------------+-------------+
|          2 | TRM55971.00  | Y           |
|          3 | TRM57971.00  | Y           |
```

Add a new row to the antenna table when needed. This is done automatically by Dataworks mirroring software. IGS antennas are named in ftp://ftp.igs.org/pub/station/general/rcvr_ant.tab.

```
describe receiver_firmware;
+--------------------+-----------------+------+-----+---------+----------------+
| Field              | Type            | Null | Key | Default | Extra          |
+--------------------+-----------------+------+-----+---------+----------------+
```

```
| receiver_firmware_id | int(5) unsigned | NO   | PRI | NULL    | auto_increment |
| receiver_name        | varchar(20)     | NO   |     | NULL    |                |
| receiver_firmware    | varchar(20)     | NO   |     | NULL    |                |
| igs_defined          | char(1)         | NO   |     | N       |                |
+----------------------+-----------------+------+-----+---------+----------------+
select * from receiver_firmware;
+----------------------+-------------------+--------------------+-------------+
| receiver_firmware_id | receiver_name     | receiver_firmware  | igs_defined |
+----------------------+-------------------+--------------------+-------------+
|                    2 | TRIMBLE NETR5     | 4.03               | Y           |
|                    3 | TRIMBLE NETR5     | 4.17               | Y           |
|                    4 | TRIMBLE NETR5     | 4.22               | Y           |
```

Add a new row to the receiver_firmware table when needed. There is a row for every receiver firmware field value, even when the receiver_name is the same as in other rows. This is done automatically when needed by Dataworks mirroring software. IGS receivers are described in ftp://ftp.igs.org/pub/station/general/rcvr_ant.tab.

```
describe radome;
+------------+-----------------+------+-----+---------+----------------+
| Field      | Type            | Null | Key | Default | Extra          |
+------------+-----------------+------+-----+---------+----------------+
| radome_id  | int(5) unsigned | NO   | PRI | NULL    | auto_increment |
| radome_name | varchar(15)    | NO   |     | NULL    |                |
| igs_defined | char(1)        | NO   |     | N       |                |
+------------+-----------------+------+-----+---------+----------------+
select * from radome;
+-----------+--------------+-------------+
| radome_id | radome_name  | igs_defined |
+-----------+--------------+-------------+
|         2 | NONE         | Y           |
|         3 | SCIT         | Y           |
|         4 | SNOW         | Y           |
...
```

Add a new row to the radome table when needed. This is done automatically as needed by Dataworks mirroring software.

```
describe metpack;
+-------------+-----------------+------+-----+---------+----------------+
| Field       | Type            | Null | Key | Default | Extra          |
+-------------+-----------------+------+-----+---------+----------------+
| metpack_id  | int(5) unsigned | NO   | PRI | NULL    | auto_increment |
| metpack_name | varchar(15)    | NO   |     | NULL    |                |
+-------------+-----------------+------+-----+---------+----------------+
select * from metpack;
+------------+--------------+
| metpack_id | metpack_name |
+------------+--------------+
|          2 | no metpack   |
+------------+--------------+
```

Add a new row to the metpack table when needed. This is done automatically as needed by Dataworks

mirroring software.

```
describe network:
+--------------+-----------------+------+-----+---------+----------------+
| Field        | Type            | Null | Key | Default | Extra          |
+--------------+-----------------+------+-----+---------+----------------+
| network_id   | int(3) unsigned | NO   | PRI | NULL    | auto_increment |
| network_name | varchar(50)     | NO   |     | NULL    |                |
+--------------+-----------------+------+-----+---------+----------------+
select * from network;
+------------+--------------+
| network_id | network_name |
+------------+--------------+
|          1 | COCONet      |
|          2 | TLALOCNet    |
+------------+--------------+
```

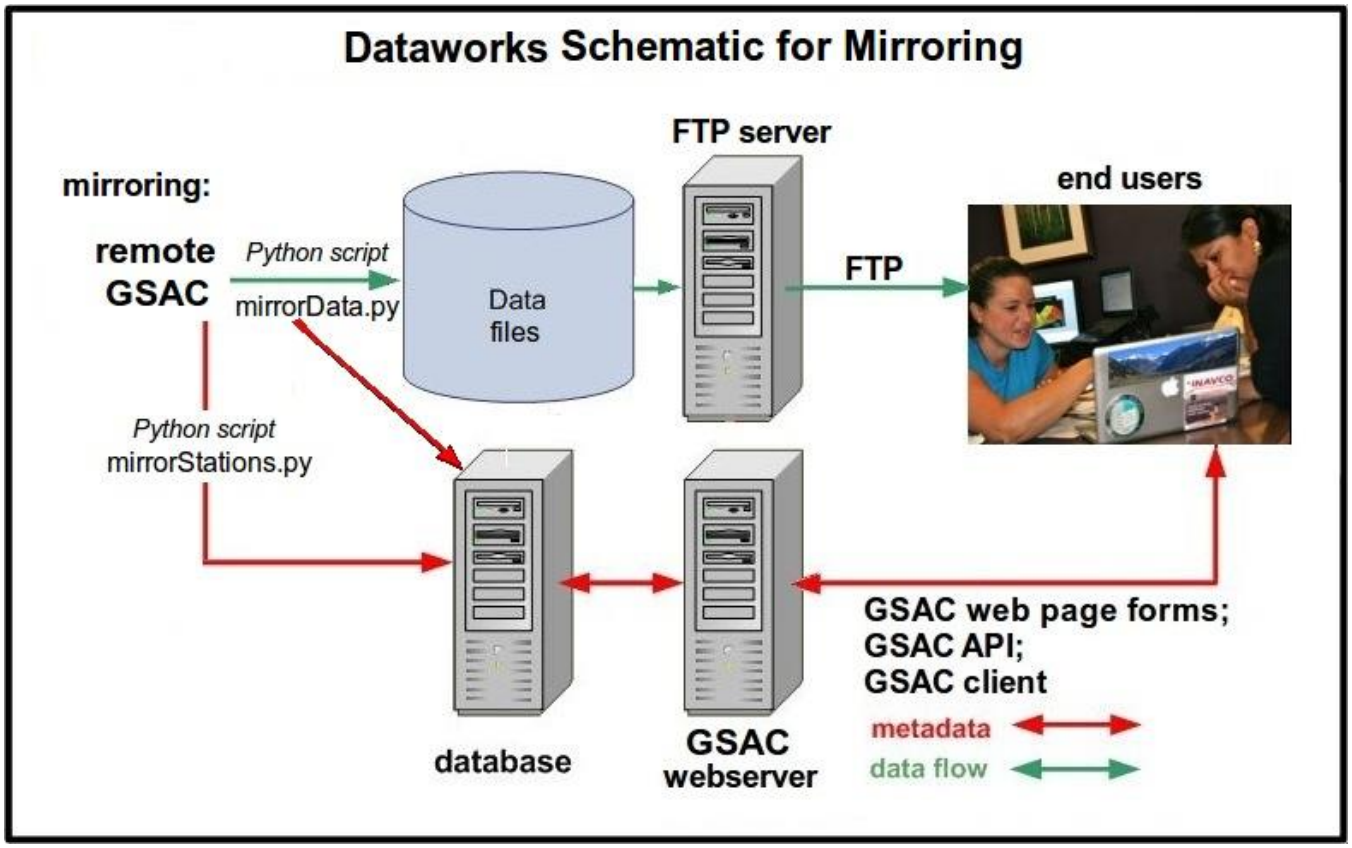
Add a new row to the network table if you add a new station in a new network.

3.3 Database Maintenance

Populating and maintaining the database for Dataworks is *necessary* to operate Dataworks and GSAC. Maintaining your database for Dataworks is your responsibility. See Appendix A for more detailed hints on working with the Dataworks database.

4. Mirroring Station and Equipment Metadata and Data Files from a Remote Data Center Running GSAC

The Dataworks Mirror Module is code that you can use to copy GNSS data files, and station and instrument metadata, from a remote GSAC-enabled data repository, such as the UNAVCO GSAC, to populate your Dataworks installation. This process is called "mirroring." Mirroring copies station and instrument information, and data files, from a station network or for a list of stations, stored at the remote GSAC repository. Mirroring is entirely optional. The schematic figure below shows the mirroring functions .



You need not mirror anything from a remote GSAC to use Dataworks. Dataworks can be used to distribute GNSS information from your GNSS instrument network using Dataworks software to retrieve GNSS instrument data (Section 5) from receivers, and using GSAC to make it available. *If you do not plan to mirror from a remote GSAC, do not do anything described in this section 4.*

For mirroring from a remote GSAC-enabled data repository, once a day two Python scripts, mirrorStations.py and mirrordata.py, are run automatically by a Linux 'cronjob'. This is described in section 4.2

While the mirroring code supplied with Dataworks initially uses the UNAVCO GSAC system (mirroring from UNAVCO), with a few small changes you can mirror from any operating GSAC. The examples here and in the Python scripts use the UNAVCO GSAC as the remote GSAC.

Note: Dataworks server setup should include creating an account for the user with the operations role, that is, the user having responsibility to ensure Dataworks day-to-day operations occur as expected. This user is designated in this documentation as having user name “ops”.

4.1 Python Script for Updating (Mirroring) the Station & Equipment Information from a Remote GSAC

The Dataworks database for GSAC has information (metadata) about stations and equipment (instruments) that you wish to provide with GSAC.

To mirror station and equipment information from a remote GSAC, Dataworks uses the `mirrorStations.py` script. The Python script file is located at

`/dataworks/mirror_station_metadata/mirrorStations.py`

The `mirrorStations.py` script:

1. during the first-ever run, `mirrorStations.py` makes the initial population of a Dataworks database with the stations in the network being mirrored, and all the equipment sessions at each station.

2. during subsequent runs, `mirrorStations.py` finds and adds newly-added stations in the network, and the equipment sessions at those stations.

3. finds and adds new equipment sessions at any existing station, in every run.

4. updates existing equipment sessions (db equip_config records), when the metadata was changed at the remote GSAC. Usually session stop_time is changed daily at active stations, to the end of day, at least in UNAVCO practice.

For `mirrorStations.py` the DGD must exist and have the correct schema before you run the script.

The Python script or program is file

`/dataworks/mirror_station_metadata_from_unavco/mirrorStations.py`

The script inserts new metadata, gathered from the remote agency, about stations, and / or about new station equipment sessions, to the existing database. This is used both to initially populate an existing empty database, and also to add new stations, or to add new equipment sessions at stations already in the database.

This script is run automatically every day by the crontab job. To see the crontab file, in account ops do command `crontab -l`.

You can also run the script by hand. The instructions shown here are in the header of the script file itself, if you need to refer to them.

(some configuration items: the line in the Python file:

```
satellite_system = "GPS"  # default  could be for example "GPS,GLONASS"
                          # or "GPS,GALILEO" or "GPS,GLONASS,GALILEO"
```

where the db field "satellite_system" is only changed when you change to receivers with more constellations. The db field "satellite_system" lists which GNSS constellations are received by your network.

And you can choose logging to screen on or off with values 1 or 2 in the Python code line:

```
logflag =1 # USE =1 for routine operations;  or use =2 to print log lines to screen
```

With Dataworks for GNSS mirroring, station data used to populate your database is gathered using the remote GSAC service, in a "GSAC full csv" formatted file. The GSAC query is part of mirrorStations.py operations. You do not run GSAC queries separately. The Python script `mirrorStations.py` does it for you.

The `mirrorStations.py` is executed with a Linux command like:

```
/dataworks/mirror_gps_data/mirrorStations.py  stationgroup  databasehost  gsac-dbacct gsac-dbpw  databasename
```

For "**stationgroup" you can u**se a network name employed in the remote GSAC archive (e.g. COCONet or another). Use local names for the database host name ('databasehost'), db account name ('gsac-dbacct'), db account password ('gsac-dbpw'), and database name ("databasename") such as Dataworks. So an actual command could be:

```
/dataworks/mirror_gps_data/mirrorStations.py COCONet localhost gsacdb gsacpw
Dataworks
```

Alternately, for 'stationgroup' you can use, for separate stations, not a network name, but inside one pair of the quotation marks " ":

a. a single station's four char ID like "POAL" in " ";

b. semi-colon separated list of four char IDs, in " " separated with *semi-colon ;* (not comma) like "p123;p456" ;

c. wildcards using the * character, like "TN*" in " ".

So for the stationgroup argument, put any or all of a. thru c. in one string with no spaces in " ", like "POAL;P12*;TN*" . The semi-colons separate the items in stationgroup argument. Upper case and lower case in station four char IDs are the same in GSAC use. Wildcards with the "*" *between* characters, like P*X, *do not work*.

So a purely hypothetical command is:

```
/dataworks/mirror_station_metadata/mirrorStations.py
"PALX;PHJX;PJZX;PLPX;PLTX;PTAX;PTEX" localhost gsacdb gsacpw Dataworks
```

to add and/or update those seven stations metadata in the dataworks database.

Use this option with caution; you can add a lot of stations not in your network to your system with one simple command.

If you make an incorrect addition, you can remove those rows from the database. First delete the incorrect new rows from the equip_config table for the incorrect stations, and then delete selected rows from the station table, with your database editor tool.

The `mirrorStations.py` script makes makes a log file named `/dataworks/logs/mirrorStations.py.log.nn`,

where nn is the day of month number (00 to 31).

Log files are overwritten once a month.

Look at the log file /dataworks/logs/mirrorStations.py.log.nn after each run. Look for errors noted in lines with the exact word PROBLEM and fix any problems.

Normal log file lines when the station (station table rows) and equipment sessions (equip_config table rows) are already in the database are like this example:

```
*******  Check station POAL
    Next equip session data set (line): POAL,POAL_TNET_MX2013,19.1187,-
98.6552,3992,shallow foundation pillar,
      ,2014-08-15T14:30:30,2014-11-
23T23:59:45,TRM57971.00,NONE,5000112724,0.5,0.0000,0.0000,TRIMBLE
NETR9,4.85,5137K78333,15,Altzomoni,,Mexico,,,,1
      Antenna name TRM57971.00 is a valid IGS name, and in the IGS file
rcvr_ant.tab
      RADOME name NONE is a valid IGS name, and in the IGS file rcvr_ant.tab
      Receiver name TRIMBLE NETR9 is a valid IGS name, and in the IGS file
rcvr_ant.tab
      This input equip_config session for station POAL has equip_config_start time=
2014-08-15T14:30:30   _stop time=_2014-11-23T23:59:45
```

```
      Already have station POAL in the database.
      Look for this equip config record in the db with sql SELECT
equip_config_id,equip_config_start_time,equip_config_
      stop_time from equip_config where station_id= 1 and equip_config_start_time=
'2014-08-15T14:30:30'
      This session # 36 is already in the db.
      and the session start and stop times match. Go try next input data line.
*******   Station POAL is up-to-date in the database (station count so far is 1)
```

When a new equip_config row is added to the database, the log file is like:

```
    Next equip session data set (line): VRAI,Veragua__CRI2012,9.9249,-
83.1906,444.28,building roof,,2014-12-03
      T00:00:00,2014-12-
08T23:59:45,TRM59800.00,SCIT,5208354391,0.0083,0.0000,0.0000,TRIMBLE
NETR9,4.85,5114K74710,15,Cano Negro,,Costa Rica,,,,,WXT520,H3130004,135
      Antenna name TRM59800.00 is a valid IGS name, and in the IGS file
rcvr_ant.tab
      RADOME name SCIT is a valid IGS name, and in the IGS file rcvr_ant.tab
      Receiver name TRIMBLE NETR9 is a valid IGS name, and in the IGS file
rcvr_ant.tab
      rcvsampInt  = _15
      metpackname = _WXT520_  metpackSN =_H3130004_   metpack_id=1
      This input equip_config sesssion for station VRAI has equip_config_start
time= 2014-12-03T00:00:00   _stop time=_2014-12-08T23:59:45
      Already have station VRAI in the database.
      Look for this equip config record in the dq with sql  ...
      SELECT equip_config_id,equip_config_start_time,equip_config_stop_time from
equip_config where station_id= 116 and equip_config_start_time= '2014-12-
03T00:00:00'
      Insert this new equipment session into the db with SQL:
      INSERT into equip_config (station_id, create_time, equip_config_start_time,
equip_config_stop_time, antenna_id, antenna_serial_number, antenna_height,
radome_id, radome_serial_number, receiver_firmware_id, receiver_serial_number,
satellite_system,sample_interval) values (116, '2014-12-09T21:42:12', '2014-12-
03T00:00:00', '2014-12-08T23:59:45',  12, '5208354391', 0.0083,  4, ' ', 15,
'5114K74710',  'GPS', 15)
     Inserted a new equipment session into the db
```

There is a summary at the end of the log file; here is a typical daily result:

```
SUMMARY of mirrorStation.py processing:
  No new stations added.
  No new equipment sessions added.
  There are 109 stations in the database.
  There were 412 station - equipment sessions matches in the database.
  Updated 66 equip config table stop times.
  Complete at Thu, 22 Jan 2015 19:10:57 +0000 UTC
```

If mirroring added a new station, you must add some station metadata by hand to the database in the "station" table, equip_config table and to other tables. These values are not supplied by a remote GSAC. Here is how:

The metadata needed is the agency_id numbers for *two* fields in the new station table row. Find the agency_id numbers in the agency table.

```
mysql> select * from agency;
+-----------+-------------------------+-------------------+
| agency_id | agency_name             | agency_short_name |
```

```
+-----------+--------------------------+--------------------+
|         1 | UNAVCO                   | UNAVCO             |
|         2 | UNAM                     | UNAM               |
|         3 | Universidad de la Sierra |                    |
+-----------+--------------------------+--------------------+
```

If the names of the agency or agencies you need are not in the agency table, add them with a mysql insert command. There are two agency_id numbers in each station table row.

Add the required agency_id numbers to the new station table row, like this:

```
update station set operator_agency_id=<AID1> where four_char_name="ABCD";
update station set data_publisher_agency_id=<AID2> where four_char_name="ABCD";
```

Substitute valid agency_id number values for the variables <AID1> and <AID2>.

If mirroring added a new equip_config row, you must add some metadata by hand to the database in the equip_config table. These values are not supplied by the UNAVCO GSAC.

The metadata needed is metpack_id, metpack serial number, and the radome serial number.

Find the equip_config_id for the new equip_config row (like 333) by looking in the mirror station log file.

Then do two commands like this, but with the correct serial numbers:

```
update equip_config set metpack_serial_number="123456789" where equip_config_id=333;
update equip_config set radome_serial_number="98765432" where equip_config_id=333;
```

Also you must add the metpack_id to the same new equip_config row. Look in the metpack table to find the metpack name in use:

```
mysql> select * from metpack;
+------------+--------------+
| metpack_id | metpack_name |
+------------+--------------+
|          2 | WXT520       |
+------------+--------------+
```

If a metpack name is not in that table, add it to the metpack table with a mysql insert command.

Note the metpack_id for the metpack name. Add that number to the equip_config row, like this:

update equip_config set metpack_id =<MN> where equip_config_id=333;

where <MN> is the correct metpack_id, an integer.

4.2 Daily Run of mirrorStations.py by Crontab

`mirrorStations.py` is run once a day by a crontab job. To see the crontab file, in account ops run the command crontab -l.

The example below shows two lines in the crontab file for account 'ops' are as below (00 10 * * * means 10:00 UTC every day):

```
00 10 * * * /dataworks/mirror_station_metadata_from_unavco/mirrorStations.py
TLALOCNet localhost dbacct dbacctpw Dataworks

15 10 * * * /dataworks/mirror_gps_data_from_unavco/mirrorData.py localhost  dbacct
dbacctpw  Dataworks 4daysback today TLALOCNet
```

The run time for mirrorStations.py is in this example10:00:00. The process will complete in about 2 minutes for 100 stations.

Note in this example crontab runs mirrorStations.py and then mirrorData.py, in the correct order, 15 minutes apart.

4.3 Handling mirrorStations.py: Catching Up

If your Dataworks has been offline or shut down for more than a day, run mirrorStations.py to catch up with the data you have not mirrored, e.g.:

```
/dataworks/mirror_station_metadata/mirrorStations.py COCONet localhost gsac-dbacct
gsac-dbpw Dataworks
```

where 'gsac-dbacct' is the database account name for GSAC, 'gsac-dbpw' is the data base account password, and 'Dataworks' is the name of the MySQL database used by GSAC. 'COCONet' is an example for a network name in the remote GSAC you are mirroring.

You then run mirrorData.py as described below.
If you run mirrorStations.py by hand, the screen output looks like

```
./mirrorStations.py COCONet localhost  dbacct dbacctpw  Dataworks
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
44313    303  44313  129k     0      0    1822       0 --:--:--  0:01:12 --:--:--  6814
```

This output is made by the GSAC request. You do not need to understand it, or to do anything with it.

4.4 Python Script to Mirror Data Files from a Remote GSAC and to Update the Database

Dataworks for GNSS can "mirror" (copy) GNSS data files from a remote GSAC with a Python script, and populate and update the Dataworks table datafile in the database.

The DGD must have complete, correct, and up to date metadata for all stations, and for all the equipment sessions at all stations, before you can successfully mirror data files. Sections 4.1 – 4.3 cover these topics.

The Python script to mirror (copy) data files from a remote GSAC is

`/dataworks/mirror_gps_data/mirrorData.py`

This script populates the DGD data files metadata (in table 'datafile'), and also copies the complete GNSS data files to your computer from the remote GSAC. You can also run the script by hand. The instructions shown here are also shown in the header of the script file itself, if you need to refer to them there.

You should run the script `mirrorStations.py` every time you run `mirrorData.py`, before you run `mirrorData.py`, to ensure changes at stations being mirrored are captured.

Before you run mirrorData.py, you must configure it. Open the file mirrorData.py in an editor; find the lines labeled “CONFIGURATION” in the header and do those steps.

"Crontab" will run this script automatically every day, typically at about 10:00 UTC. The exact time is not important. Do the Linux command crontab -l to see for example (00 10 * * * means 10:00 UTC every day):

```
00 10 * * * /dataworks/mirror_station_metadata_from_unavco/mirrorStations.py
TLALOCNet localhost  dbacct dbacctpw  Dataworks
```

```
15 10 * * * /dataworks/mirror_gps_data_from_unavco/mirrorData.py localhost  dbacct
dbacctpw  Dataworks 4daysback today TLALOCNet
```

The `mirrorData.py` command run by crontab uses the two words "4daysback today" (or "Ndaysback today") which are special input arguments to cover the recent days. See code in the Python file dealing with the exact words 'daysback' and 'today.'

Also, you can run this program by hand with commands like this, for example, to get data files for a date range you want to update, for example, for this one-month date range shown:

```
./mirrorData.py dbhost gsac-dbacct gsac-dbpw dbname 2014-04-01 2014-04-30
stationgroup
```

dbhost is like 'localhost', gsac-dbacct and gsac-dbpw are the MySQL account names and password to write to the database named "Dataworks", and stationgroup is the remote GSAC archive name for a network, like COCONet.

For a station name list in place of a network name for stationgroup, these inputs are accepted within a single pair of quotation marks "":

a) a single station's four char ID like "POAL" (in " ").

b) semi-colon separated list of four char IDs, separated with ; semicolon (not comma) like "p123;p456". The semi-colon separates the items in station group list.

c) wildcards using * like "TN*"

So for the station group, put any or all of a. thru c. in one string with no spaces, like "POAL;P12*;TN*" .

Upper case and lower case in station four char IDs are the same in GSAC use, so poal is interpreted POAL.

A hypothetical command to add data files from 7 sites is:

```
/dataworks/mirror_gps_data/mirrorData.py localhost gsac-dbacct gsac-dbpw Dataworks
4daysback today "PALX;PHJX;PJZX;PLPX;PLTX;PTAX;PTEX"
```

Before you run this, you must first do the equivalent command with mirrorStations.py for these 7 sites.

If you do wish to mirror those stations' data files routinely, simply add this one command to your crontab file for daily operation.

As an example of what to expect for time demands, with a good internet bandwidth this script takes several hours to get all GPS files from 100-some COCONet stations for one recent year. But a daily update takes only a few minutes.

The script makes makes a log file named `/dataworks/logs/mirrorData.py.log.nn`

where nn is the day of month number. Routine processing details go to the log file. Always look at the log file every day for lines marked "PROBLEM" and deal with whatever happened.

For a log file example, the top of a `mirrorData.py` log file is like:

```
    Process to search for and download gps data files at UNAVCO, from _2014-12-05_
through _2014-12-12_
    Log file describing processing by mirrorData.py
    mirrorData.py run started at 12Dec2014_17:09:06
 ***** *****  Look at the log file /dataworks/logs/mirrorData.py.log after each run.
Look for errors in lines noted with the word PROBLEM  ***** ***** *****
    Get list of sites from the UNAVCO GSAC server.  The Linux command is
```

```
    /usr/bin/curl -L  "http://www.unavco.org/data/web-
services/gsacws/gsacapi/site/search/sites.csv?output=site.csv&site.group=TLALOCNet"
 > dataworks_sites_short.csv
  UNAVCO GSAC site search query succeeded. Have site list dataworks_sites_short.csv
with 12 stations.
   1  New Station POAL:
   Get metadata about all GPS data files from station POAL in the given date range,
with linux command
    /usr/bin/curl -L  "http://www.unavco.org/data/web-
services/gsacws/gsacapi/file/search?file.sortorder=ascending&site.code=POAL&file.da
tadate.from=2014-12-
05&output=file.csv&site.name.searchtype=exact&site.code.searchtype=exact&limit=5000
&file.datadate.to=2014-12-12&site.interval=interval.normal"  > data_file_info.csv
 Count of gps datafiles for this station from UNAVCO GSAC in this time interval: 35
   station POAL; this data file's metadata from UNAVCO GSAC is
    POAL,GNSS RINEX Observation (Hatanaka  Unix
Compressed),9214946fee511fbbed4723f1e7efc360,733341,2014-12-08 00:00:00,ftp://data-
out.unavco.org/pub/rinex/obs/2014/341/poal3410.14d.Z,2014-12-07 00:00:00,2014-12-07
23:59:45,0.0,POAL_TNET_MX2013
   Load mirror the dataworks db with the gps file metadata, and download the gps
file from UNAVCO
     Already have metadata for this datafile, poal3410.14d.Z, in the db datafile
table, for station_id 1, at datafile_id =5772
    Download (or verify having) the data file from ftp://data-
out.unavco.org/pub/rinex/obs/2014/341/poal3410.14d.Z
     No problems occurred running wget command  wget -nv -c -x -nH -P /data
ftp://data-out.unavco.org/pub/rinex/obs/2014/341/poal3410.14d.Z
```

The log file ends with a summary like this:

```
Summary of mirroring data files from UNAVCO, during 2015-01-17 to 2015-01-22
 number of COCONet stations in the UNAVCO archive checked for GNSS data files
     in this time interval: 109
          obs files totalsize=     233.668  MB
          nav files totalsize=      10.672  MB
          met files totalsize=       5.873  MB
          total size all files=    250.213  MB  or     0.244 GB
          obs file count=   365
          nav file count=   338
          met file count=   257
          count of all required data files found in this time interval 960
          About new files to download:
          db: count of data files' info TO insert in the db            170
          db: count of data files' info Inserted in the db             170
          db: count of data files' info FAILED inserts in the db         0
          files: success count of data files to download from UNAVCO,
             or already downloaded:                                    952
          files: count of wget problems encountered                      5
    Completed mirrorData.py at Thu, 22 Jan 2015 19:18:38 +0000 UTC
```

Every time mirrorData.py runs, it writes a new log file `/dataworks/logs/mirrorData.py.log.nn`. Look at the log file `/dataworks/logs/mirrorData.py.log.nn` after each run. Look for errors noted in lines with the exact word PROBLEM and deal with any problems.

The error "wget returned status=2048" means there was a temporary problem getting this file from UNAVCO. In virtually all cases the same wget download will be repeated the next day for this same file, and

get it. Tries are made five days in a row. If you want to make sure, run the wget command listed in the log file on a command line.

4.5 Daily Run of mirrorData.py by crontab

`mirrorData.py` is run once a day by a crontab job.

The example below shows two lines in the crontab file for account 'ops' are as below (00 10 * * * means 10:00 UTC every day):

```
00 10 * * * /dataworks/mirror_station_metadata_from_unavco/mirrorStations.py
TLALOCNet localhost dbacct dbacctpw Dataworks

15 10 * * * /dataworks/mirror_gps_data_from_unavco/mirrorData.py localhost  dbacct
dbacctpw  Dataworks 4daysback today TLALOCNet
```

The run time for `mirrorData.py` is in this example10:15:00. The process will complete in about 2 minutes for 100 stations.

Note in this example crontab runs `mirrorStations.py` and then `mirrorData.py`, in the correct order, 15 minutes apart.

4.6 Handling mirrorData.py: Catching Up

Look at the log file `/dataworks/logs/mirrorData.py.log.nn` after each run. Look for errors noted in lines with the exact word PROBLEM and fix any problems. Every time mirrorData.py runs, it writes a new log file `/dataworks/logs/mirrordata.py.log.nn.` `nn` is the day of month number.

If your Dataworks has been offline or shut down for more than a day, run mirrorData.py to catch up with the data you have not mirrored.

First you run `mirrorStations.py` as described above. Then run `mirrorData.py` like:

```
/dataworks/mirror_gps_data/mirrorData.py localhost gsac-dbacct gsac-dbpw Dataworks
2014-12-18 2105-01-05 COCONet
```

where the date range (like "2014-12-18 2105-01-05") spans the days the mirroring was not done.

5 Dataworks Operations with Local GNSS Stations

The Dataworks Download, Ingest, and Export software enables users to create a data repository populated by data from a GNSS network managed and maintained at the local level. These modules require the DGD, both for their own successful operation and for seamless integration with the GSAC. Using these Dataworks modules enables users to create a data repository with public web services for data access, populated by data from a GNSS network managed and maintained at the local level.

The Dataworks data download and processing suite is composed of three components to help manage the workflow when downloading GNSS data files from a remote station or network of remote stations, validating the data transfer and registering each individual station's data files in the database provided as part of the software suite.

The following diagram displays the functional interaction between the processes and components of the software suite, with expanded explanations describing the performance of the manager, exporter, and ingester processes in the following sections.

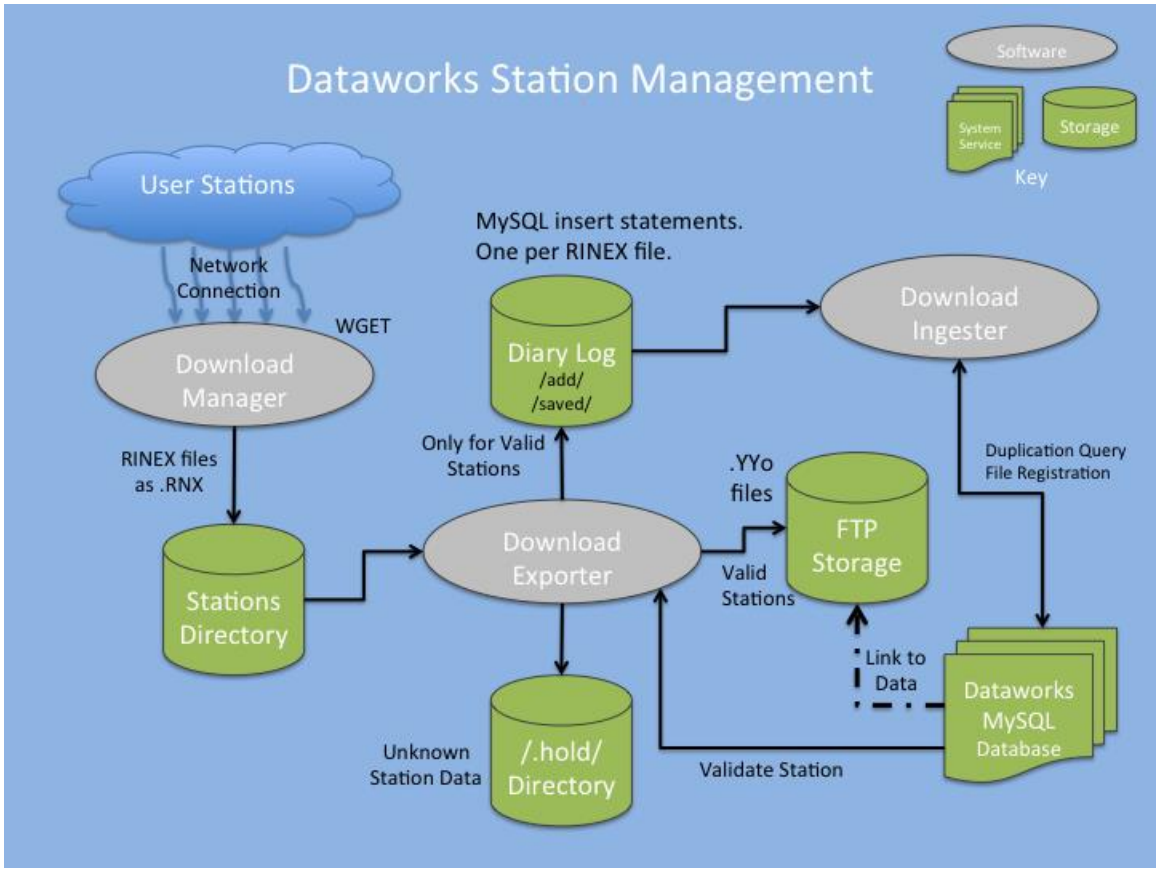


FIGURE 1.

- The downloadManager script uses unique station and receiver configurations to download files from remote receivers and store the files in a local station/data directory. An internet connection to the receiver is required.

- The downloadExporter scans station directories for completed file downloads, building a list of files to mine for meta-data in order to build a diary file. The diary file is used to register the file and associated meta-data in the dataworks database. Once the downloadExporter has built the individual diary file it moves the download file to the systems ftp storage directory so external users can retrieve the data from the server.
- The downloadIngester is responsible for scanning for new file diaries and registering those new downloads in the dataworks database to make the files available to the GSAC service.

Appendix B details a checklist of actions to complete for the initial setup of the downloadManager and necessary checks and changes required for equipment session changes.

5.1 Add a New Station and/or a New Equipment Session to the Database

To successfully add new GNSS data files from a station to your Dataworks repository, you will need to register information about the station into the database table station (as described in the database schema section of Dataworks documentation), including details about the instrumentation (receiver, dome, antenna) during the time frame the data is being collected (an "equipment session" row in the table equip_config linked to the station).

To minimize the effort and add consistency to the process of adding a new station and equipment session to the database, it is recommend users utilize the Python script and template file supplied with Dataworks:

```
dataworks/db_aids/insertNewStation.py
```

Using this script avoids having to make detailed MySQL commands for this case of database maintenance.

Complete instructions for using this script are contained in the script file's header, which a user may read by opening the file with a text editor. The steps to use this script for adding a station are as follows:

- Gather the necessary information (metadata) for establishing a new station and equipment session, most importantly the four character ID associated with the station. This value cannot duplicate the four character ID of any station already registered in the database. A simple search of the web interface can assist the user in identifying possible values.

- The user should select a station name, and collect values for the latitude, longitude and ellipsoid_height of the station. They should know the monument style (e.g. pillar, deep drilled braced, building roof) of the station, and have a date and time selected for putting the station into service. The user should also collect all the pertinent equipment data, including the antenna type, the antenna offset in height from the reference mark, the antenna serial number, the dome

type, and information about the receiver including type, serial number, firmware version and sample interval.

- Finally the user will need to provide data on the organization name, the location name and country along with the associated network name. The user may also collect information for an optional metpack (meteorological data collection) and its serial number.

With the data collected a user can now utilize a text editor to create a file containing a comma separated values in the db_aids directory, which needs to contain all the information required for completing a station insertion into the database. Multiple stations may be entered in a single file with each station separated by a blank line.

The formats for the fields in the file are as follows:

```
4_char_ID [type=string(4)], station_name [type=string(50)], latitude [double],
longitude [double], ellip_height [unit='m'], monument_description
[type=string(70)], IERSDOMES [type=string], session_start_time [type='date'
format='yyyy-MM-ddTHH:mm:ss zzzzz'],
session_stop_time [type='date' format='yyyy-MM-ddTHH:mm:ss zzzzz'],
antenna_type [type=string(15)], dome_type [type=string], antenna_SN
[type=string], Ant_dZ [unit='m'], Ant_dN [unit='m'], Ant_dE [unit='m'],
receiver_type [type=string(20)], firmware_version [type=string], receiver_SN
[type=string], receiver_sample_interval, locale_name [type=string(70)], country
[type=string(70)], agency_name [type=string(100)], network_name
[type=string(50)], metpack_name [type=string(15)], metpackSN [type=string]
```

An example file, station_equip_data.csv, has a line:

```
S123,test station 123,16.2623,61.5275,25.67,building roof,97103M001,2010-08-
27T00:00:00,0000-00-
00T00:00:00,TRM55971.00,NONE,1440911917,0.0000,0.0000,0.0000,TRIMBLE
NETR5,4.17,4917K61764,30,LES ABYMES Guadeloupe,France,Institut Geographique
National,COCONet,no metpack,,
```

NOTE: NO COMMAS or APOSTROPHES are allowed in any of the field values. Using these prohibited characters will cause an error in the insert script and will fail to load your station data into the database. Also note the use of the date 0000-00-00T00:00:00 to indicate that the station_stop_time is undefined is allowed for a new station (i.e you are actively collecting data from this station).

With the data collected and the file complete users can run the command:

```
./insertNewStations.py station_equip_data.csv localhost dbacctname dbacctpw
dbname
```

Where the first argument is the comma separated file name, the second points to the name of the local machine, the dbacctname is the name of the database account used to log into the database, along with dbacctpw for the password and finally, dbname is the name of the database, such as DATAWORKS_TEST. Information from the script process is printed on the screen and problems encountered are registered in the log file:

```
Results_dataworks_station_equip_mysql_insertions_yyyymmdd_hhmmss.txt
```

Common errors with registering station and equipment session data are missing values in mandatory fields within the csv list (only metpack data is optional), or fields in the file are created with too many characters to register in the database. In the case of errors registering a station, users should refer to the results log file for trouble shooting information and carefully check their values against the format fields to ensure correct data entry.

5.2 Downloading GNSS Station Data Files and Associated Population of the Dataworks Database

The Download Manager

The function of the download manager included as part of the Dataworks software package is to provide processing to automate remote GNSS station downloads as a data input source for database ingest which supports the archive process within Dataworks. The download manager software components are located in the dataworks primary users home directory. This file package can be accessed either directly through login on the computer platform itself or via a remotely hosted connection such as an X windows enabled terminal (xterm) using the ssh application or appropriate X hosting software for graphical interfaces. (See the Dataworks Administrators Guide for assistance with setting up an interface to the local server.)

The only receiver types currently supported by the download manager are the Trimble NetR9 and NetR5 models. To effectively establish a connection with the receiver the user should be familiar with the setup of the Trimble receiver and have access to the users manual. An electronic copy of the NetR9 receiver manual is provided in the supportDocs directory for reference purposes. Establishing a station to utilize FTP transfer and a configuration compatible with a NetRS receiver is possible such that users can download Trimble's binary format directly from the NetRS, but the additional steps needed to make these files integrate with Dataworks would be the user's responsibility. Further support for new or additional receivers, such as the NetRS, will be integrated into Dataworks on a case-by-case basis.

The download manager utilizes the wget utility to transfer files from the receiver on a user established schedule set through configuration file utilities provided in the Dataworks package. The download manager requires that the receiver can be reached by an Internet connection. As the Trimble NetR9 is a highly configurable receiver the download manager lets the user establish multiple configuration files for each receiver session setup in order to support several varieties of automatic download management. Internet connectivity provides the user the ability to connect to the Trimble receiver with an ordinary web browser to help the user establish correct receiver operation and to provide a fast and effective means of gathering setup information for configuring the automated download process.

The following screen capture is an example of using the Trimble web interface to access the receiver and highlights the data that user will need to gather from the receiver or the user manual to set up the download manager.

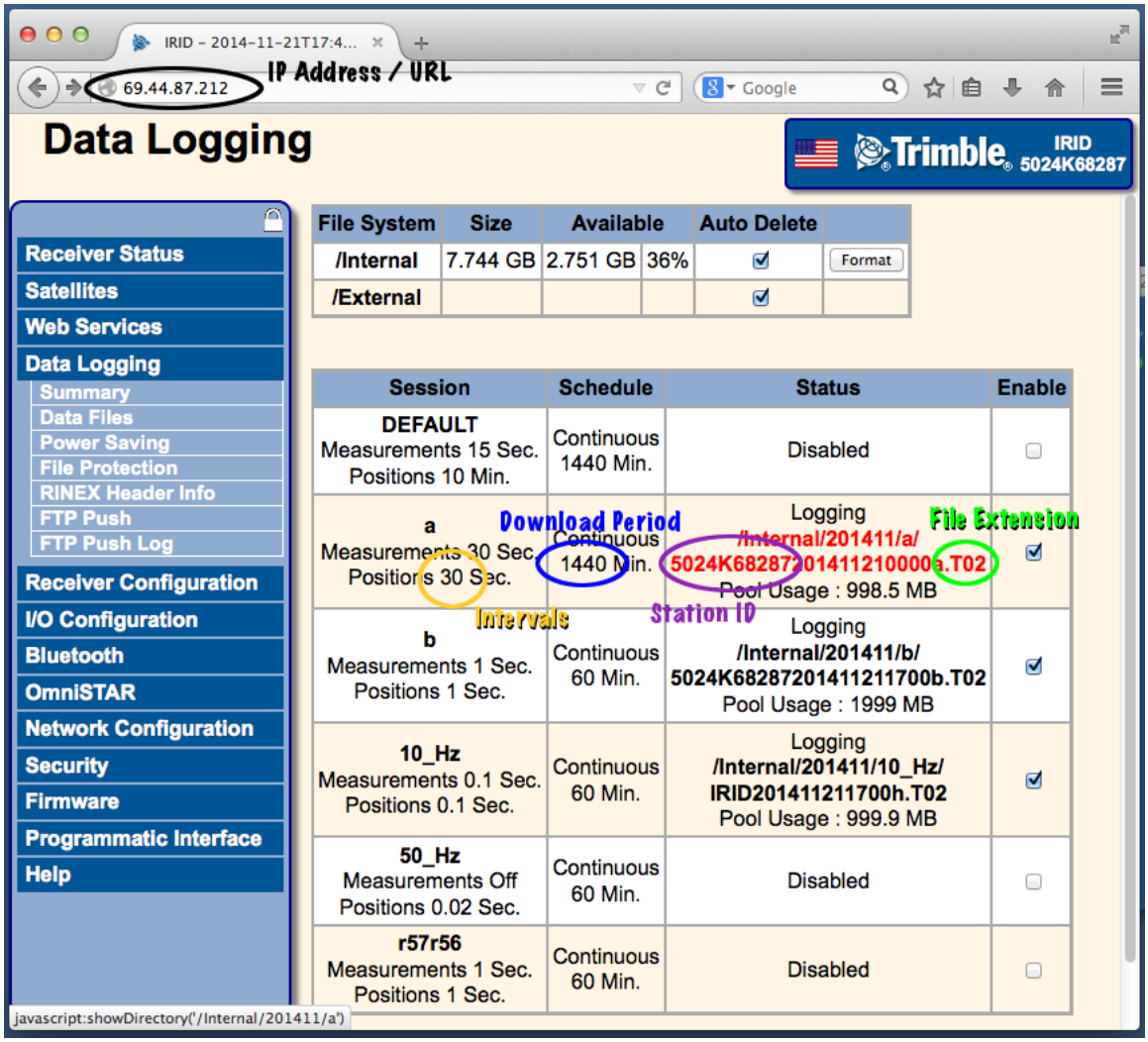

IRID - 2014-11-21T17:4...
69.44.87.212
IP Address / URL
Google
Data Logging
Trimble
IRID
5024K68287
Receiver Status
Satellites
Web Services
Data Logging
Summary
Data Files
Power Saving
File Protection
RINEX Header Info
FTP Push
FTP Push Log
Receiver Configuration
I/O Configuration
Bluetooth
OmniSTAR
Network Configuration
Security
Firmware
Programmatic Interface
Help
File System
Size
Available
Auto Delete
/Internal
7.744 GB
2.751 GB
36%
Format
/External
Session
Schedule
Status
Enable
DEFAULT
Measurements 15 Sec.
Positions 10 Min.
Continuous
1440 Min.
Disabled
a
Measurements 30 Sec.
Positions 30 Sec.
Download Period
Continuous
1440 Min.
Logging
/Internal/201411/a/
5024K68287201411210000a.T02
Pool Usage : 998.5 MB
File Extension
Intervals
Station ID
b
Measurements 1 Sec.
Positions 1 Sec.
Continuous
60 Min.
Logging
/Internal/201411/b/
5024K68287201411211700b.T02
Pool Usage : 1999 MB
10_Hz
Measurements 0.1 Sec.
Positions 0.1 Sec.
Continuous
60 Min.
Logging
/Internal/201411/10_Hz/
IRID201411211700h.T02
Pool Usage : 999.9 MB
50_Hz
Measurements Off
Positions 0.02 Sec.
Continuous
60 Min.
Disabled
r57r56
Measurements 1 Sec.
Positions 1 Sec.
Continuous
60 Min.
Disabled
javascript:showDirectory('/Internal/201411/a')

FIGURE 2a.

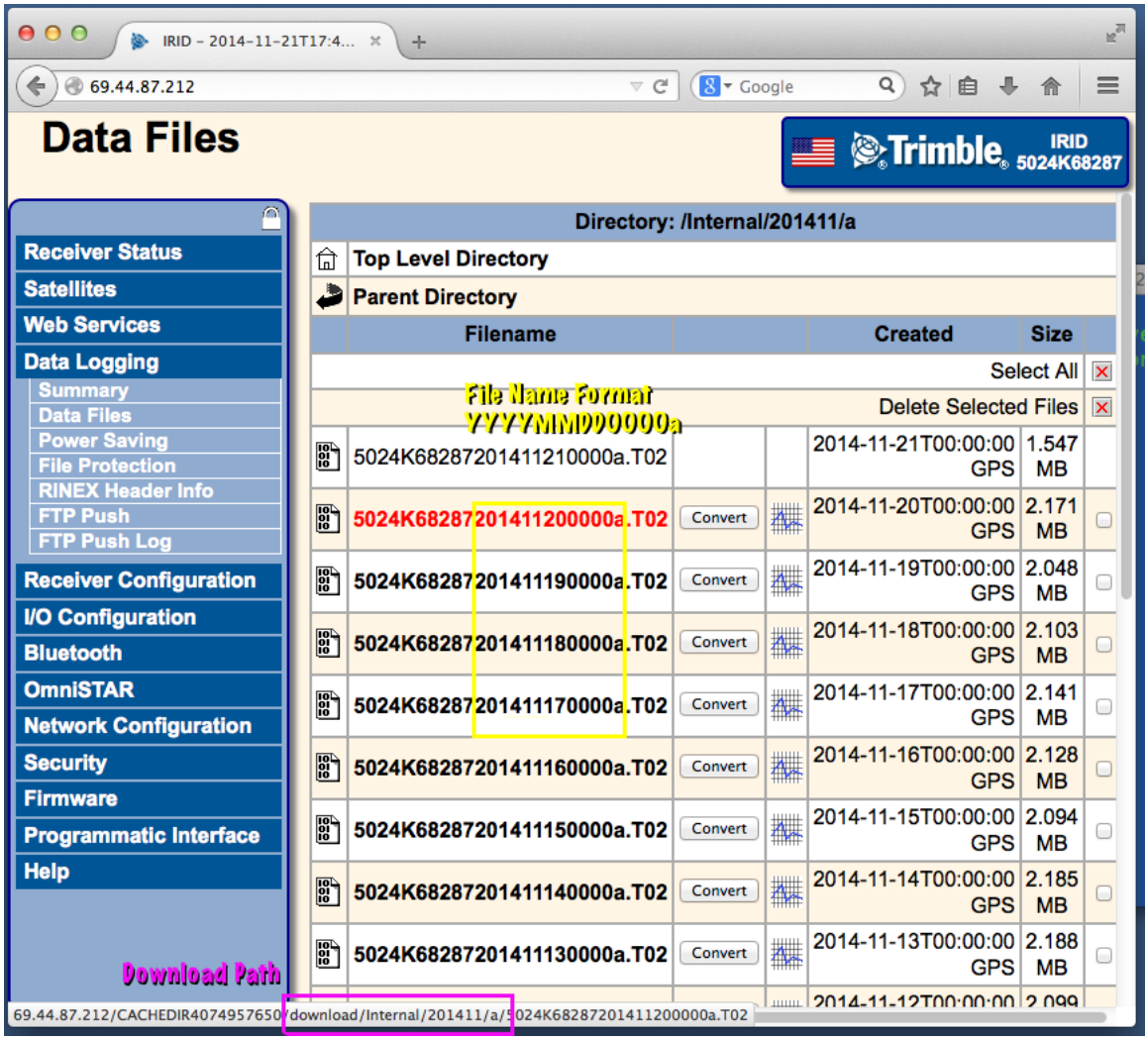


FIGURE 2b.

In the next portion of this users manual the reader can follow along with the steps to setup a receiver type and station download schedule, both necessary steps to complete before activating the download manager script. Once these configurations are correctly established the download manager can run continuously to retrieve station files by a prioritized schedule, which will be covered further when discussing the download managers operation.

The first step to the setup operation requires the user to log into the Dataworks server (using the xterm software and ssh utility to connect to the server or directly logging in via the hardware console. NOTE use ssh –Y if your remote console is configure to use X windows forwarding to provide graphical interfaces).

Once logged in to the server, navigate to the dataworks/lib directory to begin manager setup. The library contains utility scripts used to assist the user to customize the configuration of the download manager. The first step is to configure your receiver settings using the script changeReceiverType.py. This is a python script that assists you in correctly configuring the download requirements for a Trimble NetR9 receiver session.

Typing the command:

```
changeReceiverType.py --help
```

Will bring up the help feature for the command:

```
usage: changeReceiverType.py [-h] [-vis] [-update] [filename]

Used to add/update station receiver configuration sets to dataworks
functionality

positional arguments:
  filename    filename REQUIRED for use with update option

optional arguments:
  -h, --help  show this help message and exit
  -vis        use the graphical interface for input.
  -update     update the file configuration content.
```

The user has options to start a graphic interface with the –vis option (provided X hosting is configured correctly) or may navigate a text interface if the option flag is not set. The user also has the option of using the script to change a previous receiver configuration by setting the –update flag. If the user doesn't know the file name they wish to update, a full set of available files you may modify are stored in the library directory under the conf directory (dataworks/lib/conf). The use of the -update flag without a matching receiver configuration file will produce an error and repeat the help message.

Using the graphic interface produces a form to fill in with the appropriate receiver session information, which correctly configures the receiver type for download manager to access the desired file sets. The following image shows an example based of the configuration parameters defined in the previous figure.

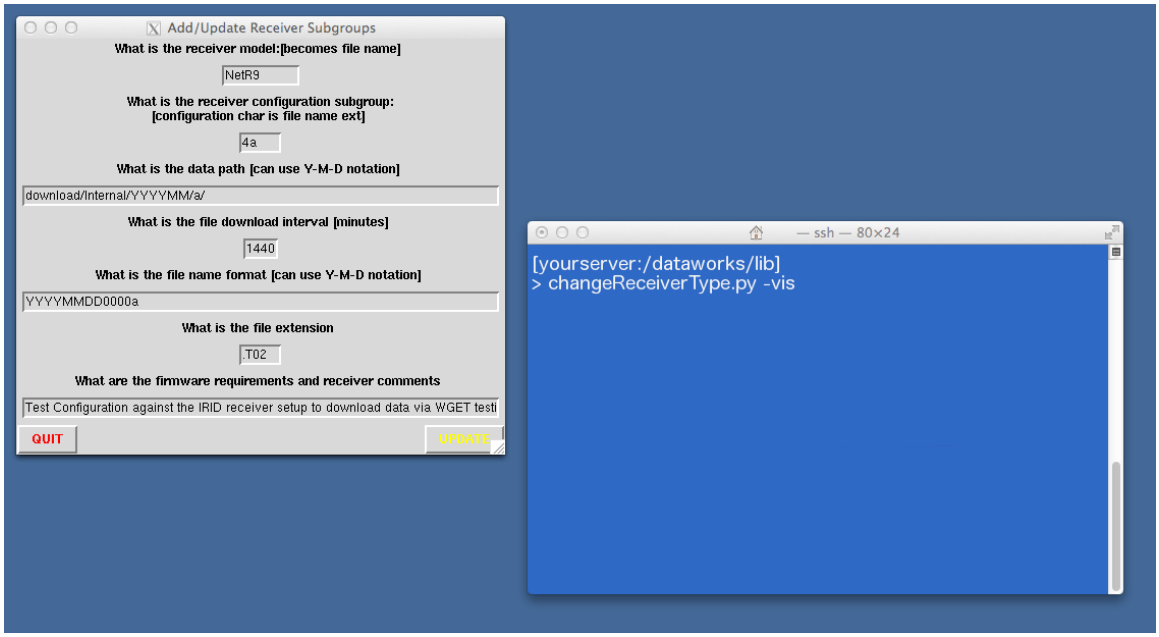


FIGURE 3.

Users will add the appropriate data to the form or text fields to complete a receiver session configuration. The receiver model, a sub group descriptor for the session, the data path as documented by the receiver manual or directly from the web interface to the receiver, the desired download interval for data in minutes, the file format, and the file extension are all required inputs. The user may also add comments to the comments field to describe the session configuration and reasons that these data values were utilized.

In our example above using the Trimble web interface information we can use the displayed values to set up our receiver session settings. Notice that the receiver setup uses numeric replacement characters to set a download pattern for the receiver. In the example above the numerical year is replaced with the substitution notation YYYY, the numerical month with the substitution notation MM, the numerical day with substitution DD and so on. In our case since the file is meant for daily download (1440 minutes) the hour and minute characters are the specific digits used in the file name on the receiver. If the receiver is providing updates on an hourly basis and using letters to differentiate files the special case replacement string cidx can be used and will be replaced by the characters [a-x] depending on the hour of the day (cidx means character index).

The download manager uses pattern substitution to create the file specific name from this template before it attempts to download data from the receiver. It is critical you get the pattern correct so the manager can use the pattern matching as expected to download the current data set from the receiver. If a user wants to use more frequent file download intervals such as hourly then the pattern matching template would use the characters HH for digit replacement in the template, and use the characters mm if they were to use a higher download rate for receiver sessions that output files at the mm minute interval. For data sets at a resolution more frequent than 30-minute intervals it would be a best practice to find another retriever solution such as wget to connect to the receiver and capture streaming data into a file.

Once a user clicks the update button a configuration file is established for the receiver session. To help keep configuration of receiver types from becoming confused only one receiver session may be added at a time. The use of update will continue to change information for that receiver session only. Additionally once update is clicked the first time, changing the first two fields of the form will no longer have any effect on establishing a new receiver session. The user must quit and start the program again if they want to add or change another receiver session type.

Using the command changeReceiverType.py without any options will bring up text prompts to add the same information to create a receiver session configuration file. For removing old configuration files users may utilize a command line option in an xterm or work through a file manager to navigate to the dataworks/lib/conf directory and delete the desired configuration file(s). Manual deletion methods are deliberate to keep from wiping out configuration files through an accidental click of a mouse.

Once the user has established one or more receiver session configurations they are ready to set up station download configurations. This interface and command works in a similar manner to the receiver configuration.

Typing on the command line:

```
changeStation.py --help
```

This will bring up the help features for the command:

```
usage: changeStation.py [-h] [-vis] [-update] [staID]

Used to add/update station configuration sets to dataworks
functionality

positional arguments:
  staID       UNIQUE Station ID REQUIRED for use with update option

optional arguments:
  -h, --help  show this help message and exit
  -vis        use the graphical interface for input.
  -update     update the file configuration content.
```

Every receiver configured to record data is set up with a unique station ID or a receiver serial number as show in figures 2a & 2b. This unique identifier is utilized to create a specific station configuration file to collate data storage. This identifier will be used to create a RINEX file storage directory for each station inside the dataworks/stations/ directory. This identifier must also correspond to a station ID (four character ID) correctly registered in the database, as described in the previous section. In the instance where the user utilized the receiver serial number as the station ID the configuration permits the use of an alias which maps the receiver serial number to the four character ID used as the primary identifier in the dataworks database.

Using the `-vis` flag can bring up a station entry form where you fill in the data necessary to uniquely identify the station, set its receiver session type, fill in its Internet address and its

download priority and define the number of days of data you will want to retrieve. Also if the receiver is set up with authentication required you will enter the username and password needed to download data. If you want to learn more about the authentication function of your receiver please see the receiver user manual.

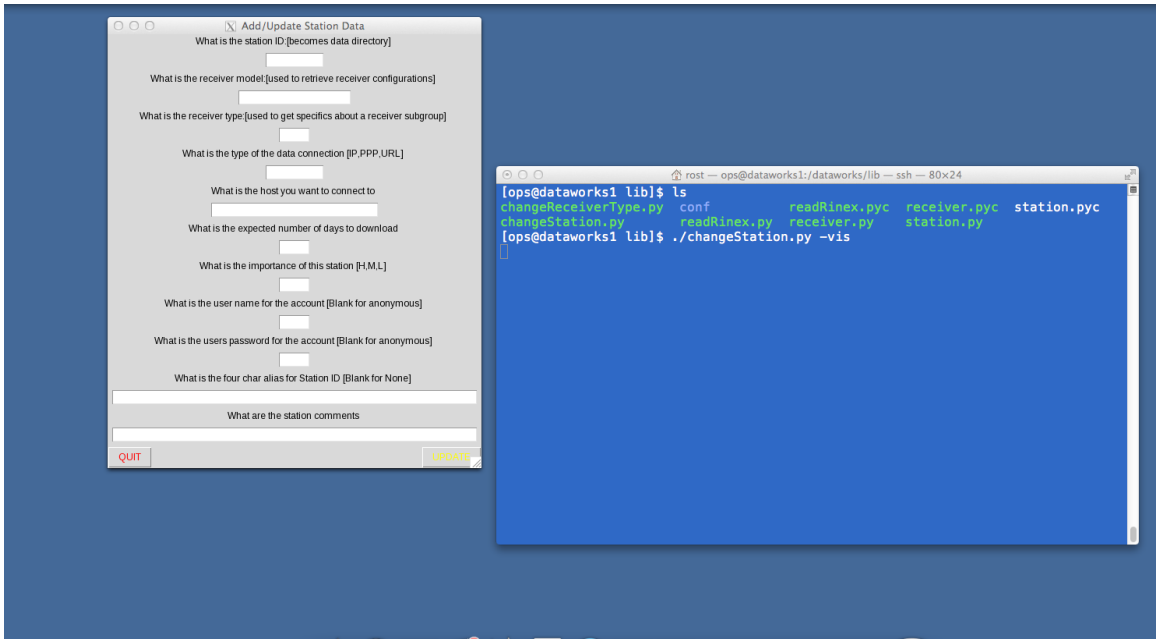


FIGURE 4.

Again using the –update flag will set the configuration for a single station as defined by the station id and store it in the stations directory as previously discussed. The current download manager only uses the IP or URL connection type to download data via the http protocol. In its current configuration the download manager will use either type to indicate the http method of transfer and doesn't discern between the connection types at this time. Future versions of the download manager have the potential to implement upgrades for PPP or an FTP connection for data downloads at a date yet to be determined.

Also, as with adding a receiver, using the command changeStation.py without any command arguments will run the text version of the form for new station entry. If you wish to remove a station permanently from the downloader you can delete the stationID directory and all its contents from the dataworks/stations directory. As before removal of stations is a manual process by decision, and requires the user to either use command line options or a file manager to remove those stationID's from the download manager's configuration.

With the configuration files and directories for one or more stations set, the download manager is capable of using these files to automatically retrieve the stations according to priority and schedule. When started the download manager scans the stations directory and creates a list of all stations that have the download type of IP or URL. Setting a station to a data type besides URL or IP ensures that the station will not be included in the

download, a convenient feature for station management if a user needs to suspend downloads from a station for a period of time.

After a prioritized download list is created the download manager scans the stationID directory inside the stations directory. It checks for the existence of a [stationID].active file in the directory indicating that a download manager is already collecting data for that station. It also checks for a [stationID].last file to determine the last time data was collected from the station. Using the configuration file and the last successful download file, the download manager builds a list of files for the receiver format template based on the current date and time in order to retrieve current data from the receiver and place in the [stationID]/data directory.

Once the targeted file list is built, the download manager starts the utility wget as a separate process to the download manager script. The wget utility is responsible for independently managing the data download from one receiver at a time. The wget utility keeps a log file in the data directory for each download reporting its overall progress. When wget starts a download it labels the download file [filename_from_template].tmp to indicate that the file is incomplete. When wget signals it has completed its attempt to download a file, the download manager will validate the file is complete and then rename it to [filename_from_template].CRX or [filename_from_template].RNX to indicate a complete formatted file was downloaded from the receiver. The current configuration of the download manager is set to default to request HATANAKA compressed RINEX version 2.11 from the Trimble NetR9 receiver. Changing the appropriate parameters in the downloadManager configuration files allows a user to download standard RINEX 2.11 data formats. Additional download capabilities may be added with furture updates to the software.

After the RINEX files are downloaded successfully and validated as complete, they are removed from the download list and the last successful scheduled download file is updated by the manager. If a user is having problems downloading data from a receiver at a specific station they should check for a [filename_from_template].wget.log file in the specific stations data directory (dataworks/stations/[stationID]/data) to search for clues regarding issues with downloading data.

The download manager in Dataworks is capable of controlling how many downloads to individual stations it will run at one time and the user can tune this value by changing settings in the download manager configuration file. The configuration file is a JSON (Java Script Object Notation) formatted file, which can be read by a text editor and modified to suit the users preferences. If the server has a limited bandwidth users may wish to tune this number to a lower setting to test download performance. If the process of downloading stations takes longer than desired, the user may wish to increase the number of simultaneous downloads to experiment with decreasing download times and improving data access. Setting this value to a number higher than the number of stations you need to download from will have no effect on improving download times.

The following line describes the default setting for the download manager configuration file:

```
{"compact": true, "dlPause": 60, "maxProcess": 5, "terminate": false,
"aliveReport": 20, "dlType": ["IP", "URL"]}
```

The first parameter controls the type of file download. Setting compact to true indicates the user desires to download files as compact RINEX to match the data format provided by the Dataworks mirroring capabilities from the UNAVCO archive. Setting the value to false will download the files in standard RINEX 2.11 format and other pieces of the management software will store them uncompressed in the Dataworks ftp archive.

In the case that the download manager has passed out a controlled number of downloads to wget it will pause and wait for those downloads to finish before it requests further downloads. This dlPause configuration determines how long this wait period holds before the manager resumes it's processing again. The period of one minute is a default value but the user may tune the pause time based on station configuration. If the download manager is retrieving files once a day you may wish to increase the pause time to keep the manager inactive for a longer period of time. A long pause time will mean that when the manager is on a rest cycle it will not respond to stop requests until that cycle is over. As a guide its best practice to keep the pause period to a time about 1/5th to 1/10th the time between file downloads to keep the manager responsive, the log files compact, and the data timely, while the system overhead remains low. The aliveReport variable controls how often the downloadManager makes a log entry to update the software status. By default with a one-minute pause time and 20-cycle aliveReport the software will update the log entry about once every 20 minutes to indicate that it is still working.

The maxProcess value determines how many stations will get downloaded at one time, and as mentioned can be tuned according to your bandwidth availability and number of stations. The dlType sets a value used in the configuration files to determine the types of downloads supported by the manager as was also mentioned earlier. The final value of the configuration file terminate, is a switch used by the download manager between scans of the station directory to determine if the manager should stop downloading files and shutdown the manager software.

The download manager and its supporting scripts and configuration files are found in the dataworks/ops directory (ops is short for operations, the activities at the center of the Dataworks data download manager). The primary python script that starts the download manager is dlManager_start. This script checks to ensure no active download managers are running by querying the operating system for processes that may have been left running by an unexpected system change. This check prevents a user from starting more than one manager and prevents contention for system resources and potential data corruption. This start up script is also responsible for clearing any [stationID].active files that may remain from an abnormal exit, before it sets the downloadManager.conf files terminate variable to false and starts the downloadManager.py script (the primary software component of the download manager).

Users should refrain from running or stopping the downloadManager.py script directly to maintain data coherence. Use of the dlManager_start and dlManager_stop scripts are recommended to ensure all data downloads are completed, validated, and processed correctly. Both start and stop scripts can be called with a –debug flag if the user desires extra file logging to help with troubleshooting file downloading. Be warned that since the download manager starts wget as a separate and independent process the wget logs are in a separate location and also download manager reporting is not synchronized to a single

process. As file downloads take an asynchronous amount of time, out of order log entries can be generated unless you tune the manager to a single download process.

The best practice method to start the download manager is to run the command line:

```
./dlManager_start &
```

The use of the & makes this command starts the manager as a users background process reporting its activities to the log file downloadManager.py.log in the dataworks/logs directory. All log files for the downloader software are rotated once weekly with the last 5 logs stored in the logs directory.

The dlManager_stop script will take care of stopping the manager and its wget processes once an individual file download is completed. This means that if wget is in the middle of a long file download process the download manager may continue to run for some time until that download completes. If the download manager enters a rest cycle it will not complete a shutdown process until the rest cycle completes. Additionally once the stop script is ran the download manager will not start any further station downloads while giving the current file downloads a chance to complete. The point is to ensure users resist the temptation to kill the download process until they ascertain it is absolutely necessary, as this leads to an inconsistent shutdown state for the download manager.

When in doubt a user should examine the log files carefully to determine if the download manager has stopped responding to input. If you have run the stop script but see no indication that the manager has stopped queuing further downloads and you don't see a log entry to diary that the manager has received a terminate signal, then you can feel confident that the download manager has stopped responding.

If the system is ever shut down unexpectedly or a case arises where you must kill the download processes; it's considered a best practice once you have made certain to terminate all downloadManager.py processes to run the dlManager_stop script once to completion before using the dlManager_start script to restart the download manager package.

The Download Exporter

The function of the download exporter software included as part of the Dataworks package is to provide processing to extract information from station downloads and prepare them for insertion into the data ingest for archive purposes. The download exporter software components are located in the dataworks home directory. As with the manager this file package can be accessed either directly on the computer platform itself or via a remotely hosted connection such as an X windows enabled terminal (xterm) using the ssh application or appropriate X hosting software for graphical interfaces. The software comprising the primary components of the Dataworks system will then be found in the dataworks/ops directory. As with the manager, there you will find the start and stop scripts and the configuration files to change the variable behaviors of the software.

All the configuration files are JSON formatted and can be edited with a text based file editor to tune the system to meet the users requirements. The configuration for the download exporter contains the following information:

```
{"xpPause": 60, "maxProcess": 10, "terminate": false, "aliveReport":
20, "ftpRoot": "/data/", "ftpPrefix": "ftp://dataworks1/"}
```

The xpPause variable defines how long the export process will wait idle before checking to see if there is any new data available for export. The download exporter scans the dataworks/stations/[stationID]/data directory for files with a .CRX or .RNX extension. Files with this designation are validated files from the download manager ready for export to the database. When it finds the appropriate type of files the export manager places a [stationID].expLock file in the dataworks/stations/[stationID]/ directory to indicate that it is processing the downloaded files so it can block other processes from moving that data while it is being exported.

It is possible to use the exporter services to support other datasets within the Dataworks structure. If a user choses to manually download file sets from other receivers and use their own set of processes to convert the data over to either RINEX 2.11 or Compact RINEX 2.11 formats the user can process this data as the would an automatically downloaded station. All they would have to do is use the changeStation.py script to generate a station with a suitable configuration file, then copy the data into the data directory for the station with the appropriate file name and extension. The downloadExporter would then treat that data as though it had been retrieved by the downloadManager and export the data set to the ftp server while creating a diary entry for adding the new files to the Dataworks database.

The maxProcess value defines the maximum number of processes the user would like to utilize for allowing the download exporter to read, validate and mine the download data. This variable will also determine the number of database connections used to access data values in the database. The aliveReport and terminate variables are used in a similar manner to the value explained for the download manager. Terminate signals the export process to finish its file processing, clear it's locks and gracefully exit processing. If the value is true then the downloadExporter.py script should in most instances end its processing on the sever in the time specified by xpPause. The aliveReport value in conjunction with the xpPause determine how often the process makes a log entry while it is waiting to process more data.

The values for ftpRoot and ftpPrefix let the user customize where they want the ftp services on the Datworks server to store data. The ftpPrefix is used to register the server's hostname as it will be accessed through the network configuration in relation to the server. Since it is possible to use multiple configurations of ftp on a host with different name service registrations the user is responsible to set this file to the appropriate configuration.

The ftpRoot location sets the destination for archiving the downloaded files in the archiving directory structure. As users can add, expand or change disks they can direct the export manager to store the files in the appropriate location as it relates to the local storage tree configuration by changing this variable. The current configuration of the Dataworks server is to use the /data/pub directory of the local disk space to store files in a predefined file structure below that storage area. System changes using a path structure not meeting this

standard configuration may require internal code changes in the downloadExporter.py script and can impact the transfer, registration and availability of products to the Dataworks services.

With the configuration file and directories for the server set, the download exporter is capable of using these values to automatically process the station specific RINEX files. Once it builds a process list and sets the appropriate lock files, the manager starts sub processes to scan the files in the station directory to begin mining them for meta-data.

One of the first steps to the data mining is to determine if the [stationID] variable assigned to the data files is registered in the database. To achieve this goal the export process must connect to the MySQL database using database scripting and a correct configuration file. As this file is a common configuration file to both the exporter and ingester functions it is kept in the library area of the Dataworks services vice the ops directory.

Users may access and make appropriate configuration changes by modifying the file dbaccessor.conf in the dataworks/lib/conf/ directory, the same directory that houses the unique configuration files for the receiver setups. Like all the configuration files this one is formatted as a JSON file, and is a standard text file, which can be edited with any favorite application.

The following is an example of a default configuration for MySQL access:

```
{ "user": "dataworks1", "password": "1dwdbdwdb1", "host": "127.0.0.1",
"database": "DATAWORKS_TEST"}
```

The values set for user, password, host, and database reflect the settings used for logging into the MySQL service which is the key support service to the database in the software suite and should be provide as part of the users default settings once the server is delivered. When security considerations require users to change passwords this file will need to be modified to reflect changes to the users password.

Without proper database access, or when stations are downloaded without having a properly registered four character id in the database, the export manager will give up processing those files since there is no way to correctly register them in the database without key information retrieved from the database schema. The process to modify a text file and use a python script to properly add a new station to the database was covered in the first section of this chapter. Any data sets that are not successfully validated against the database are moved into a storage area as a stop hold point until the database configuration is corrected to accept their processing.

These stop hold files can be found in the dataworks/.hold directory (note the '.' in front of the hold which in terms of the operating system designates this as a hidden directory unless the user explicitly tries to list all directories). It is a good practice to scan this directory once in a while to determine if there are a build up of files indicating that something in your station or database configuration has changed and needs to be investigated further.

The export manager will contact that database to validate station existence and look up information about the stations id number and equipment configuration id number. Once it

is established that the station is properly registered for archive storage, the meta-data extracted from the file is used to register the GNSS data into the local database. The mining process scans the downloaded RINEX file, specifically it reads the file to determine the start and stop time of the file and the epoch intervals. Additionally it builds an MD5 digest to identify the file a unique set of data, and determines the file size. After the exporter has successfully mined the appropriate information about the file, it creates a registration diary file. The content of this file is a MySQL query intended to add the data file to the database via a separate registration (ingest) process.

However, once created the diary will not be used to directly register the GNSS data files into the database. The export manager stores this query diary as a text file in the log directory in the location `dataworks/logs/diaries/add/[YYYY]/[DDD]/`

The file is saved with a specific GNSS data file format with a file name composed of the four char id, numerical day of the year, series number, a .[YY]o or .[YY]d.Z extension depending on the type of file downloaded from the receiver and finally a .sql extension.

The reason for creating this diary process is two fold. If the data processing for the export manager is interrupted then using a diary minimizes the chances of corrupting the database. Again as mentioned a separate process is used to read these diaries and do the registration in the database so as to keep the access times and database utilization to a minimum. The second reason is if something does happen to the database to corrupt its contents the diary files can be reprocessed simply by copying them from an archive back into the dataworks/logs/diaries/add/ structure. This keeps the user from having to reprocess all the GNSS data files to mine the necessary elements from a RINEX file to register it in the database (ie you won't have to process terabytes of data to rebuild a database).

Once the diary process has been completed the download exporter will use a safe move procedure to transfer the file from the station directory to the ftpRoot directory. To describe the safe move procedure it is composed of a two-step process. The file is first copied from the station directory to the ftpRoot directory where it is stored in the same format as described for a diary entry except it won't have the .sql extension. Next when the copy is completed the original downloaded file is removed from the station data directory. Once all .RNX files are processed out of the directory the .expLock file is removed to signal the download exporter process that it may scan for new files the next time it runs.

This method of diary and transfer is utilized as a best practice to reasonably ensure that the GNSS data file is available in the ftp server area before the data is exposed and made available via the html interface, so a user accessing the web interface to retrieve data is able to complete a successful ftp download.

As with the download manager the download exporter is started and stopped by python scripts called dlExporter_start and dlExporter_stop. These scripts manage the necessary error checking and cleanup functions to keep the export manager running effectively and efficiently.

Again it is best practice to have users refrain from running or stopping the downloadExporter.py script directly to maintain data coherence. Use of the start and stop

scripts are highly recommended to ensure all diaries and transfers are completed, validated, and processed correctly. Both start and stop scripts can be called with a –debug flag if the user desires extra file logging to help with troubleshooting the export process. Also note as with the case of the download manager, that since the export manager starts separate and independent processes diary creation and file transfer take an asynchronous amount of time, and out of order log entries can be generated unless you tune the manager to a single download process.

The best practice method to start the download manager is to run the command line:

```
./dlExporter_start &
```

This command starts the manager as a users background process reporting its activities to the log file downloadExporter.py.log in the dataworks/logs directory. As mentioned before the log files for the software are rotated once weekly with the last 5 logs stored in the logs directory.

As before dlExporter_stop script will take care of stopping the manager and its sub processes. The mining, diary and transfer processes all go quickly but the user may still have to wait one full rest cycle before the stop script responds. The point is to ensure users resist the temptation to kill the export process until they ascertain it is absolutely necessary, as this leads to an inconsistent shutdown state for the download exporter.

As always when in doubt a user should examine the log files carefully to determine if the export manager has stopped responding to input. If you have run the stop script but see no indication that the manager has stopped queuing further processes and you don't see a log entry to diary that the exporter has received a terminate signal, then you can feel confident that the download exporter has stopped responding.

If the system is ever shut down unexpectedly or a case arises where you must kill the export processes; it's considered a best practice once you have made certain to terminate all downloadExporter.py processes to run the dlExporter_stop script once to completion before using the dlExporter_start script to restart the export manager package.

The Download Ingester

The function of the download ingester software included with the Dataworks package provides processing to add diary entries created by the exporter into the database for utilization by the web services. The download ingester software components are also located in the dataworks home directory. As with the manager and ingester, this file package is accessed in the same directory and utilizes similar start, stop and configuration files.

Worth mentioning again, all the configuration files are JSON formatted and can be edited with a text based file editor to tune the system to meet the users requirements. The configuration for the download ingester contains the following information:

{"maxProcess": 10, "terminate": false, "aliveReport": 20, "ingPause": 60}

The ingPause variable defines how long the ingest process will wait idle before checking to see if there are any new diaries for export and the aliveReport value controls how often the script reports its running status in the log file.

The download ingester scans the dataworks/logs/diaries/add/ directory tree for files with an .sql extension. Because it is scanning a tree, it builds a path list of multiple diary files sorted by year, day and station collecting all data diaries created by the exporter since the last time the export and ingest processes ran. Existing diary files are validated as having complete RINEX files from the download manager waiting in the ftp area, ready for export to the database. When the ingester finds diary files the software places an ingest.lock file in the add/ directory to indicate that it is processing available diaries and blocking other processes from moving that data while it's being ingested.

The maxProcess value defines the maximum number of processes the user would like to utilize for allowing the download ingester to communicate with the database. The terminate variable is used in a similar manner to the value explained for the download manager. It signals the ingest process to finish its file processing, clear its locks, clean up empty directories and gracefully exit processing. If the value is true then the downloadExporter.py script should end its processing on the server in the time specified by ingPause.

With the configuration file for the server set, the download ingester is capable of using these values to automatically process the diaries. Once it builds a process list and sets the appropriate lock files, the manager starts sub processes to scan the diary files in the station to prepare the sql statements for database insertion.

The processes use the database configuration file in the same manner as the exporter to connect to the MySQL database. Using key information from the diary file the ingester checks the db to ensure that it isn't going to duplicate data should the user be trying to reprocess old diary files. Once a db search verifies there is no duplication the sql statement is ran to register the GNSS data file in the Dataworks database.

When the db confirms the registration, the ingester will then move the diary file by the safe copy method described in detail earlier, placing it into a dataworks/logs/diaries/saved/[YYYY]/[DDD]/ directory. Users should be aware that the file tree will continue to expand in this directory as the download manager continues to collect files. It is recommended that users of the Dataworks suite develop their own methods for data management for this directory. See the appropriate section in the previous part of this users guide for more information on this topic of data archiving.

As with the other processes the download ingester is started and stopped by python scripts called dlIngester_start and dlIngester_stop, which manage the necessary error checking and cleanup functions to keep the export manager running effectively and efficiently. Again it is also best practice that users refrain from running or stopping the downloadIngester.py script directly to maintain data coherence. Use of the start and stop scripts are always recommended to ensure all diary registrations are completed correctly using the MySQL db utilities to help with error checking and correction. As with the other processes both start and stop scripts can be called with a –debug flag if the user desires extra file logging to help with troubleshooting the export process.

This software also uses independent processes for diary registration, which take an asynchronous amount of time, and out of order log entries can be generated unless you tune the manager to a single download process.

The best practice method to start the download manager is to run the command line:

./dlIngester_start &

As usual this command starts the manager as a users background process reporting its activities to the log file downloadIngester.py.log in the dataworks/logs directory which will be rotated weekly with the last five logs saved for reference.

As before dlIngester_stop will take care of stopping the manager and its sub processes. The registration processes all go quickly depending on db loading but the user may still have to wait one full rest cycle before the stop script responds. Always ensure users resist the temptation to kill the ingest process until they ascertain it is absolutely necessary, as this leads to an inconsistent shutdown state for the download manager.

To reiterate one last time, when in doubt a user should examine the log files carefully to determine if the download manager has stopped responding to input. If you have run the stop script but see no indication that the manager has stopped queuing further processes and you don't see a log entry to diary that the exporter has received a terminate signal, then you can feel confident that the download exporter has stopped responding.

If the system is ever shut down unexpectedly or a case arises where you must kill the ingest processes; it's considered a best practice once you have made certain to terminate all downloadIngester.py processes to run the dlIngester_stop script once to completion before using the dlIngester_start script to restart the ingest manager package.

Further technical support requests for issues not described in this users manual can be emailed to UNAVCO at the email address dataworks@unavco.org

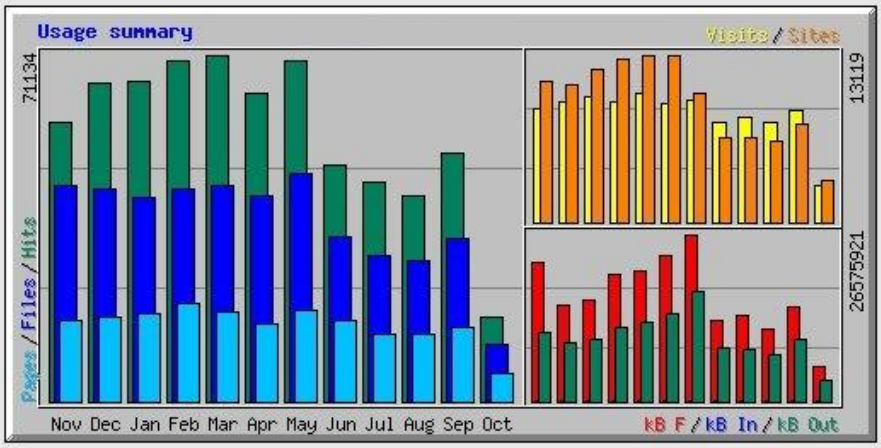
6 Metrics: Counting Data File Downloads and GSAC Requests in Dataworks for GNSS

6.1 Counting GNSS Data File Downloads by Remote Users

Dataworks for GNSS works with Webalizer to measure and report FTP downloads of GPS data files by remote users from its server.

Webalizer results are shown in public web pages. To see the summary of ftp traffic, view the Webalizer URL at a server, like http://dataworks.mycenter.org/usage/index.html. Each Dataworks for GNSS agency will have its own distinct domain, replacing that "dataworks.mycenter.org." The Webalizer web pages at a Dataworks for GNSS agency will have the same domain as the GSAC domain at that agency. Here is a sample Webalizer summary table (not an actual Dataworks for GNSS site):

Also, there is also a separate page for each month of operation (click on a month name in



Summary by Month

Month	Daily Avg				Monthly Totals							
	Hits	Files	Pages	Visits	Sites	kB F	kB In	kB Out	Visits	Pages	Files	Hits
Oct 2014	1747	1168	561	284	3227	5452528	7264	3459669	2847	5613	11681	17476
Sep 2014	1701	1114	503	292	7717	15121915	21124	10030146	8776	15099	33448	51031
Aug 2014	1361	934	449	252	6343	11487359	16396	7371508	7835	13930	28978	42203
Jul 2014	1447	963	450	266	6662	13726001	18619	8319288	8251	13964	29870	44861
Jun 2014	1616	1128	550	261	6661	12900738	24074	8508465	7838	16520	33841	48482
May 2014	2256	1505	609	310	10173	26575921	27470	17459434	9613	18909	46685	69956
Apr 2014	2112	1406	528	309	13109	23081360	23210	14069582	9271	15869	42192	63368
Mar 2014	2294	1433	590	324	13119	20730634	40235	12484960	10064	18293	44424	71134
Feb 2014	2492	1552	714	338	12726	20169745	37846	11885682	9471	19999	43460	69789
Jan 2014	2116	1351	583	316	12032	16191641	24707	9916053	9801	18091	41904	65607
Dec 2013	2103	1403	558	305	10835	15216178	27493	9261406	9458	17300	43493	65213
Nov 2013	1910	1474	556	297	11038	21991111	21153	10972095	8926	16700	44222	57305
Totals						**202645130**	289590	**123738288**	**102151**	**190287**	**444198**	**666425**

the summary table). Each month page has several tables with much information.

For measuring GPS data file download activity, only a few values are useful: Files, KBytes, and Sites. Files or Total Files (a number) shows how many files were downloaded with the FTP server by month, day, or hour. KBytes, kB, or Total KBytes is the total size in KB of all files downloaded in a month, day, or hour. Sites are counts of IP addresses requesting downloads. One or more users may come from one site. The top 30 sites (IPs) making the largest number of FTP requests is listed in a table on each month page. Those IPs are listed; you can use software tools to resolve most IPs to their owner or agency. A total count of sites by country is shown at the bottom of the page, but not all IPs give a country. Much of the information on a Webalizer month page is not particularly useful to measure GPS file FTP download activity, such as Pages. To make a useful report about GPS file download activity you will need to find relevant information in each month page, or in the overall summary page, and make your own report.

Webalizer is configured by editing `/etc/webalizer.conf` and updated by a crontab job, controlled by the file `/etc/cron.daily/00webalizer`. Webalizer reads the FTP log file `/var/log/xferlog`, and Webalizer updates files in `/var/www/usage/`. Everything shown in Webalizer results is derived from values in the FTP log, `/var/log/xferlog`.

There is a description of Webalizer on your system. On a command line enter "man webalizer" to see it. See also the Webalizer web site, http://www. webalizer.org/. The description of Webalizer at http://en.wikipedia.org/wiki/Webalizer is good.

6.2 Counting Requests for Information from GSAC by Remote Users

To measure and report the number and type of GSAC requests for information received by your Dataworks for GNSS data repository GSAC service, use the Python script Dataworks_GSAC_use_metrics.py. In account ops, use the command:

```
/dataworks/metrics/Dataworks_GSAC_use_metrics.py
```

This reads the Apache Tomcat log file `/var/log/tomcat6/catalina.out` on your system.

The results are listed on the screen. You can redirect the results to a file by the command

```
/dataworks/metrics/Dataworks_GSAC_use_metrics.py > GSAC_usage_report
```

The results look like this:

```
  GSAC Use at tlalocnet1, from 2014-11-02 to 2014 Nov 24  18:31:45 UTC

    Total count of GSAC requests   = 10618  (includes page visits as
well as actual GSAC search requests.)
    Count of site searches         = 10353   (has site/search in GSAC
API request)
    Count of site form searches    =   218   (has site/form in GSAC API
request)
    Count of file searches         =    14   (has file/search in GSAC
API request)
    Count of file form searches    =    33   (has file/form in GSAC API
request)
```

```
  Output types requested:
    site.csv                          8092
    siteops.xml                       1647
    site.xmllog                        528
    site.html                           28
    site.gsacxml                        20
    gsacxml                             17
    file.download (Webstart)             6
    site.snx                             5
    file.html                            5
    sitefull.csv                         4
    site.kmz                             2
    station.info                         2
    file.url                             1
    file.gsacxml                         1
    file.wget                            1

60 distinct sites (IPs) made GSAC requests.

The IPs making requests were:
   IP            request count   host name
   69.44.86.173          7647     moray.unavco.org
   69.44.86.88           1636     brick.unavco.org
   10.234.1.145           612      unagi.int.unavco.org
   218.29.102.114          36     hn.kd.ny.adsl
   202.56.13.99            23     (no host name found)
   132.239.153.107         19      moenkopi.ucsd.edu
    ... [more]...
```

(this is not a real Dataworks for GNSS GSAC use report.)

The 'Output types' beginning with 'file.' pertain to different ways to get information about GNSS data files which *can* be downloaded. The users making these requests *may not have* downloaded any files. *None of the values in this report of GSAC use indicate any count of FTP data file downloads.* GSAC provides answers to requests for metadata *about* GPS data files, but *not any GPS data files* themselves. FTP file download counts are shown in the Webalizer results (section 6.1).

7 Recommendations for Backup and Recovery

The critical information to backup is the FTP data directories and the MySQL database. Keep in mind, if your organization is a using the GSAC mirroring function, all the site metadata and data will be available at the originating GSAC for rebuilding the mirror.

If your organization is handling local site metadata and data with Dataworks, your data and metadata must be backed up if you wish to recover from a server or storage failure. You should at minimum backup the data and metadata directories-

The FTP data directory is `/data`.

The MySQL data directory is `/var/lib/mysql`

UNAVCO also recommends you backup any changes to GSAC code, database schemas, python code and support scripts.

If your organization already has experience and procedures for backing up file systems and MySQL databases, we recommend you follow those procedures.

If your organization has an operational data center or server facility, and you want additional recommendations for securing a back up of the FTP data directories and MySQL database, we recommend you use a network based backup utility, e.g., http://www.emc.com/data--protection/ networker.htm, and backup the critical file systems for the FTP data and MySQL data directories.

If your organization does not have a server facility and not much experience with network based backup and recovery, you will need to setup a couple of scripts to dump the MySQL database and rsync the output and the FTP data directories to another server. An example mysqldump command (see http://dev.mysql.com/doc/refman/5.1/en/mysqldump.html) looks like the following:

```
$> mysqldump -h localhost -u dataworks -p --add-drop-database
--add-droptable --single-transaction --add-locks --triggers
name_of_database
./name_of_database_dump_file-date.sql
```

More examples of mysql use are in Appendix A.

An example rsync command (http://www.comentum.com/rsync.html) looks like the following:

```
$> rsync –arv /data user@192.168.0.10:/backups
```

8 The Dataworks for GNSS Email Group and Finding Help

8.1 Dataworks for GNSS Email Forum

Join UNAVCO's Dataworks for GNSS email forum or mailing list, to receive news about dataworks and to write to UNAVCO about Dataworks for GNSS. You can receive and send emails to this list to ask questions about Dataworks for GNSS, and to exchange ideas about Dataworks for GNSS with other Dataworks for GNSS users on the list. You will be notified of new software releases, bug reports, and so on.

To subscribe to the dataworks email forum, and for general information about the mailing list, see http://postal.unavco.org/mailman/listinfo/dataworks.

To request help from UNAVCO about Dataworks for GNSS, email the Dataworks for GNSS mailing list (see just above).

*8.2 **Web Resources for Dataworks for GNSS and GSAC***

Dataworks for GNSS: http://www.unavco.org/software/data-management/dataworks/dataworks.html

GSAC: http://www.unavco.org/software/data-management/gsac/gsac.html

Known GSAC servers http://www.unavco.org/software/data-management/gsac/repositories/repositories.html

COCONet Project at UNAVCO: http://www.unavco.org/projects/major-projects/coconet/coconet.html

TLALOCNet Dataworks GSAC site:
http://tlalocnet.udg.mx/tlalocnetgsac/

Sponsor Acknowledgments

Dataworks for GNSS development was funded by NSF through the COCONet Cooperative Agreement.

GSAC, then called GSAC-WS, was a NASA ROSES ACCESS Program funded project, Cooperative Agreement NNX10AF07A (2010-2012).

In 2012 and 2013 NSF funded GSAC development at UNAVCO in support of COOPEUS.

Appendix A: Maintaining the Database and the Using MySQL Command Line Client "mysql"

Maintaining the database for Dataworks is necessary to operate Dataworks and GSAC. Maintaining your database for Dataworks is outside of UNAVCO operations, and is your responsibility. Data archives require regular attention to large and small details if they are to operate correctly. Operating a data archive with public search and access is far more demanding than keeping files on a computer disk.

In routine Dataworks operations you may need to add information to the database, and possibly correct errors. This section gives an overview of working with the database, and some resources available for doing these tasks.

Dataworks uses a MySQL database, using a schema named "Dataworks." To maintain the database you can use "**mysql" t**he MySQL command line tool; see http://dev.mysql.com/doc/refman/5.5/en/mysql.html. Or you can use MySQL Workbench, a GUI to manage MySQL databases; see http://www.mysql.com/products/workbench/.

Some basics about working with a MySQL database are in the MySQL Tutorial at:

http://dev.mysql.com/doc/refman/5.5/en/tutorial.html,

and in many other web sites, such as:

http://oak.cs.ucla.edu/cs144/projects/mysql/

https://kb.ucla.edu/articles/mysql-resources

https://www.digitalocean.com/community/tutorials/a-basic-mysql-tutorial

In this appendix we assume you know about basic database concepts, such as tables, rows (records), and fields (columns). Here are *few* examples about using mysql to see what is in a database, to enter new field values, and to change field values.

MySQL databases have accounts with names and passwords to get into mysql. Accounts have different levels of privileges or 'permissions' in MySQL, which can limit what you can do. Some accounts are only to read from the database (GSAC uses a read-only mysql account); others can add new data; others may create new accounts and change permissions as well as read and write.

You need to have an account to use mysql.

To start the mysql command line tool:

```
> mysql -h localhost -u dbacct1 -p
```

Enter password:

The mysql prompt is "mysql> ". Every mysql command ends with ";".

To exit mysql, type quit; :

```
mysql> quit ;
Bye
```

In a "mysql> " session on your terminal, to list the available databases:

```
mysql> show databases;
+--------------------------+
| Database                 |
+--------------------------+
| information_schema       |
| Dataworks                |
| ...
```

Some of the databases listed are used only by mysql itself.

You next choose one database name to work with, in most cases the database named "Dataworks":

```
mysql> use Dataworks;
```

To list the tables in that database:

```
mysql> show tables;
+--------------------+
| Tables_in_Dataworks |
+--------------------+
| access             |
| agency             |
| antenna            |
| country            |
| datafile           |
| datafile_type      |
| ellipsoid          |
| equip_config       |
| locale             |
| metpack            |
| monument_style     |
| network            |
| radome             |
| receiver_firmware  |
| station            |
| station_status     |
| station_style      |
+--------------------+
```

To see the "description" (some of the schema) for one table, 'station':

```
mysql> desc station;
+-------------------------+----------------+------+-----+---------+----------------+
| Field                   | Type           | Null | Key | Default | Extra          |
+-------------------------+----------------+------+-----+---------+----------------+
| station_id              | int(6) unsigned | NO   | PRI | NULL    | auto_increment |
| four_char_name          | char(4)        | NO   |     | NULL    |                |
| station_name            | varchar(50)    | NO   |     | NULL    |                |
| latitude_north          | double         | NO   |     | NULL    |                |
| longitude_east          | double         | NO   |     | NULL    |                |
| height_above_ellipsoid  | float          | NO   |     | NULL    |                |
| installed_date          | datetime       | NO   |     | NULL    |                |
| retired_date            | datetime       | YES  |     | NULL    |                |
| style_id                | int(3) unsigned | NO   | MUL | NULL    |                |
| status_id               | int(3) unsigned | NO   | MUL | NULL    |                |
| access_id               | int(3) unsigned | NO   | MUL | NULL    |                |
| monument_style_id       | int(3) unsigned | NO   | MUL | NULL    |                |
| country_id              | int(3) unsigned | NO   | MUL | NULL    |                |
| locale_id               | int(3) unsigned | NO   | MUL | NULL    |                |
| ellipsoid_id            | int(1) unsigned | NO   | MUL | NULL    |                |
| iers_domes              | char(9)        | YES  |     | NULL    |                |
| operator_agency_id      | int(3) unsigned | YES  | MUL | NULL    |                |
| data_publisher_agency_id | int(3) unsigned | YES  | MUL | NULL    |                |
| network_id              | int(5) unsigned | NO   | MUL | NULL    |                |
| station_image_URL       | varchar(100)   | YES  |     | NULL    |                |
| time_series_URL         | varchar(100)   | YES  |     | NULL    |                |
+-------------------------+----------------+------+-----+---------+----------------+
```

Note each line above shows the allowed data type for each field. You cannot put a text string (like 99.234 W") in "latitude," or more than 50 characters in "station_name."

To see all the values in 3 rows in the station table:

```
mysql> select * from station limit 3;

+------------+----------------+-----------------+----------------+----
-----------
| station_id | four_char_name | station_name     | latitude_north |
longitude_east | height_above_ellipsoid | installed_date      |
retired_date | style_id | status_id | access_id | monument_style_id |
country_id | locale_id | ellipsoid_id | iers_domes | operator_agency_id
| data_publisher_agency_id | network_id | station_image_URL
| time_series_URL |
+------------+----------------+-----------------+----------------+----
-----------+
|          1 | POAL           | POAL_TNET_MX2013 |        19.1187 |
-98.6552 |                   3992 | 2014-08-15 14:30:30 | NULL
|         1 |          1 |          2 |                  1 |          1 |
1 |            1 |            |                  2 |
2 |          1 | http://www.unavco.org/data/gps-
gnss/lib/images/station_images/POAL.jpg | NULL            |
```

```
|          2 | TNAM           | TNAM_TNET_MX2014 |        20.5357 |
-103.9668 |                1226.78 | 2014-09-04 00:00:00 | NULL
|         1 |         1 |         2 |                 2 |          1 |
2 |            1 |            |                   1 |
1 |          1 | http://www.unavco.org/data/gps-
gnss/lib/images/station_images/TNAM.jpg | NULL            |
|          3 | TNCM           | TNCM_TNET_MX2014 |        19.4982 |
-105.0448 |                  86.01 | 2014-09-08 17:00:15 | NULL
|         1 |         1 |         2 |                 2 |          1 |
3 |            1 |            |                   1 |
1 |          1 | http://www.unavco.org/data/gps-
gnss/lib/images/station_images/TNCM.jpg | NULL            |
+------------+----------------+------------------+----------------+----
-----------
```

3 rows in set (0.00 sec)

To see specific values from all the rows in a table:

```
mysql> select station_id, four_char_name,station_name,
latitude_north,longitude_east from station;

+------------+----------------+--------------------------+-------------
---+----------------+
| station_id | four_char_name | station_name             |
latitude_north | longitude_east |
+------------+----------------+--------------------------+-------------
---+----------------+
|          1 | POAL           | POAL_TNET_MX2013         |
19.1187 |       -98.6552 |
|          2 | TNAM           | TNAM_TNET_MX2014         |
20.5357 |      -103.9668 |
|          3 | TNCM           | TNCM_TNET_MX2014         |
19.4982 |      -105.0448 |
|          4 | TNCU           | CuauhtemocTN2014         |
28.4506 |       -106.794 |
|          5 | TNHM           | hermosilloTN2014         |
29.0813 |      -110.9703 |
|          6 | TNMR           | TNMR_TNET_MX2014         |
18.2885 |      -103.3455 |
|          7 | TNMS           | TNMS_TNET_MX2014         |
20.5347 |      -104.7967 |
|          8 | USMX           | Universidad de la Sierra |
29.8217 |       -109.681 |
|         11 | TNNX           | TNNX_TNET_MX2014         |
17.4076 |       -97.2239 |
+------------+----------------+--------------------------+-------------
---+----------------+
9 rows in set (0.00 sec)
```

To see specific values from a specific row in a table, selected by a unique value in a row

```
mysql> select station_id, four_char_name,station_name ,
latitude_north,longitude_east,installed_date,monument_style_id from
station where four_char_name="TNAM";

+-----------+--------------+----------------+--------------+--------------+-----------
--------+-----------------+
| station_id | four_char_name | station_name     | latitude_north | longitude_east |
installed_date      | monument_style_id |
+-----------+--------------+----------------+--------------+--------------+-----------
--------+-----------------+
|          2 | TNAM           | TNAM_TNET_MX2014 |        20.5357 |      -103.9668 | 2014-09-04
00:00:00 |                2 |
+-----------+--------------+----------------+--------------+--------------+-----------
--------+-----------------+
1 row in set (0.00 sec)
```

The equip_config table:

```
mysql> desc equip_config;

+-------------------------+-----------------+------+-----+---------+---
-------------+
| Field                   | Type            | Null | Key | Default |
Extra          |
+-------------------------+-----------------+------+-----+---------+---
-------------+
| equip_config_id         | int(6) unsigned | NO   | PRI | NULL    |
auto_increment |
| station_id              | int(6) unsigned | NO   | MUL | NULL    |
|
| create_time             | datetime        | NO   |     | NULL    |
|
| equip_config_start_time | datetime        | NO   |     | NULL    |
|
| equip_config_stop_time  | datetime        | YES  |     | NULL    |
|
| antenna_id              | int(3) unsigned | NO   | MUL | NULL    |
|
| antenna_serial_number   | varchar(20)     | NO   |     | NULL    |
|
| antenna_height          | float           | NO   |     | NULL    |
|
| metpack_id              | int(3) unsigned | YES  | MUL | NULL    |
|
| metpack_serial_number   | varchar(20)     | YES  |     | NULL    |
|
| radome_id               | int(3) unsigned | NO   | MUL | NULL    |
|
| radome_serial_number    | varchar(20)     | NO   |     | NULL    |
|
| receiver_firmware_id    | int(3) unsigned | NO   | MUL | NULL    |
|
| receiver_serial_number  | varchar(20)     | NO   |     | NULL    |
|
| satellite_system        | varchar(20)     | YES  |     | NULL    |
```

```
|
| sample_interval         | float           | YES  |     | NULL    |
|
+-------------------------+-----------------+------+-----+---------+---
-------------+
16 rows in set (0.00 sec)
```

To show 4 rows:

```
mysql> select * from equip_config limit 4;

+-----------------+------------+---------------------+----------------
---------+
| equip_config_id | station_id | create_time         |
equip_config_start_time | equip_config_stop_time | antenna_id |
antenna_serial_number | antenna_height | metpack_id |
metpack_serial_number | radome_id | radome_serial_number |
receiver_firmware_id | receiver_serial_number | satellite_system |
sample_interval |
+-----------------+------------+---------------------+----------------
---+
|              24 |          8 | 2014-11-15 00:17:41 | 2014-07-15
14:22:45     | 2014-12-03 23:59:45    |          2 | 5343354887
|          0.0083 |          2 | J0540019              |         2 |
|                     1 | 5341K46185             | GPS              |
15 |
|              35 |          2 | 2014-11-15 00:17:41 | 2014-09-04
00:00:00     | 2014-11-24 23:59:45    |          2 | 5343354885
|          0.0083 |          2 | K2630028              |         2 |
|                     1 | 5250K40670             | GPS              |
15 |
|              36 |          1 | 2014-11-15 00:17:41 | 2014-08-15
14:30:30     | 2014-12-03 23:59:45    |          1 | 5000112724
|             0.5 |          2 | N/A                   |         1 |
|                     1 | 5137K78333             | GPS              |
15 |
|              37 |          3 | 2014-11-15 00:17:41 | 2014-09-08
17:00:15     | 2014-12-03 23:59:45    |          2 | 5330354725
|          0.0083 |          2 | K2630031              |         2 |
|                     1 | 5250K40772             | GPS              |
15 |
+-----------------+------------+---------------------+----------------
--+
```

A lookup table example; the description:

```
mysql> desc antenna;
+-------------+----------------+------+-----+---------+---------------+
| Field       | Type           | Null | Key | Default | Extra         |
+-------------+----------------+------+-----+---------+---------------+
```

```
| antenna_id   | int(3) unsigned | NO   | PRI | NULL    | auto_increment |
| antenna_name | varchar(15)     | NO   |     | NULL    |                |
| igs_defined  | char(1)         | NO   |     | N       |                |
+--------------+-----------------+------+-----+---------+----------------+
```

the rows:

```
mysql> select * from  antenna;
+-----------+-------------+-------------+
| antenna_id | antenna_name | igs_defined |
+-----------+-------------+-------------+
|          1 | TRM57971.00  | Y           |
|          2 | TRM59800.00  | Y           |
|          3 | ASH701945B_M | Y           |
|          4 | TRM55971.00  | Y           |
+-----------+-------------+-------------+
```

To change a string value in a row in table equip_config:

```
update equip_config set radome_serial_number="SA001" where equip_config_id=30;
```

To change a numerical value in a row in that table:

```
update equip_config set antenna_height=0.00830 where equip_config_id=30;
```

To update a value in the station table for one station:

```
update station set operator_agency_id=12 where four_char_name="ST12";
```

where 12 is the id key number for the agency for station "ST12", selected from the agency table.

To find the agency id numbers, see all the rows in the lookup table for agencies:

```
mysql> select * from agency;
+-----------+-------------------------------------------------+-------------------+
| agency_id | agency_name                                     | agency_short_name |
+-----------+-------------------------------------------------+-------------------+
|         2 | Institut Geographique National                  |                   |
|         5 | University of Puerto Rico, Mayaguez             |                   |
|         8 | Suprema Corte De Justicia                       |                   |
|         9 | National Geodetic Survey                        |
...
+-----------+-------------+-----------------------------------------------------+
```

If a new agency is not yet in the database, add its name into the database agency table with this command:

```
insert into agency (agency_name) value ("Institute Castradal");
```

A similar command may be used to insert a new row into any of the lookup tables.

To change a field in a table:

```
mysql> update agency set agency_short_name="NASA" where agency_id=11;

Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0
```

To see how many gps files you have:

```
mysql> select count(*) from datafile;
+----------+
| count(*) |
+----------+
|   217532 |
+----------+
```

To see the database information about 5 gps files from some station whose station_id=12 :

```
mysql> select * from datafile where station_id=12 limit 5;
+-------------+------------+-----------------+----------------+--------
---------------+------------------+----------+
| datafile_id | station_id | equip_config_id | datafile_name  |
original_datafile_name | datafile_type_id | sample_interval |
datafile_start_time | datafile_stop_time  | year | day_of_year |
published_time      | size_bytes | MD5                              |
URL_path                                      |
+-------------+------------+-----------------+----------------+--------
---------------+------------------+------------+
|        6750 |         12 |              71 | abvi0010.14d.Z |
abvi0010.14d.Z         |                2 |               0 | 2014-01-
01 00:00:00 | 2014-01-01 23:59:45 | 2014 |           1 | 2014-01-02
00:00:00 |     580373 | 0406123a1273f13b420aa660cee2cc50 |
ftp://coconet1/rinex/obs/2014/001/abvi0010.14d.Z |
|        6751 |         12 |              71 | abvi0010.14n.Z |
abvi0010.14n.Z         |                3 |               0 | 2014-01-
01 00:00:00 | 2014-01-01 23:59:45 | 2014 |           1 | 2014-01-02
00:00:00 |      31275 | 02a84b055fb00b61a3cfec5992e4d2b8 |
ftp://coconet1/rinex/nav/2014/001/abvi0010.14n.Z |
|       79139 |         12 |              71 | abvi3250.13d.Z |
abvi3250.13d.Z         |                2 |               0 | 2013-11-
21 00:00:00 | 2013-11-21 23:59:45 | 2013 |         325 | 2013-11-22
00:00:00 |     572827 | 0a6966904f7f0a54734521aa89c56adf |
ftp://coconet1/rinex/obs/2013/325/abvi3250.13d.Z |
|       79140 |         12 |              71 | abvi3250.13n.Z |
abvi3250.13n.Z         |                3 |               0 | 2013-11-
21 00:00:00 | 2013-11-21 23:59:45 | 2013 |         325 | 2013-11-22
00:00:00 |      30967 | 603683339b362c543c3f9dd82d071c63 |
ftp://coconet1/rinex/nav/2013/325/abvi3250.13n.Z |
```

```
|       79141 |         12 |              71 | abvi3260.13d.Z |
abvi3260.13d.Z         |               2 |               0 | 2013-11-
22 00:00:00 | 2013-11-22 23:59:45 | 2013 |         326 | 2013-11-23
00:00:00 |     576469 | d5d047818ebe6c80540bd46e315495ef |
ftp://coconet1/rinex/obs/2013/326/abvi3260.13d.Z |
+-------------+-----------+----------------+---------------+--------
---------------+-----------------+
```

To select data file rows by time and at one station:

```
select * from datafile where station_id=12 and
datafile_start_time>"2013-11-28" and datafile_stop_time<"2013-11-30" ;

+-------------+-----------+----------------+---------------+--------
----------+
| datafile_id | station_id | equip_config_id | datafile_name  |
original_datafile_name | datafile_type_id | sample_interval |
datafile_start_time | datafile_stop_time  | year | day_of_year |
published_time      | size_bytes | MD5          |
URL_path                                       |
+-------------+-----------+----------------+---------------+--------
---------+
|       79155 |         12 |              71 | abvi3330.13d.Z |
abvi3330.13d.Z         |               2 |               0 | 2013-11-
29 00:00:00 | 2013-11-29 23:59:45 | 2013 |         333 | 2013-11-30
00:00:00 |     571000 | fbca32cdb4d0051970473539c10d6bf4 |
ftp://coconet1/rinex/obs/2013/333/abvi3330.13d.Z |
|       79156 |         12 |              71 | abvi3330.13n.Z |
abvi3330.13n.Z         |               3 |               0 | 2013-11-
29 00:00:00 | 2013-11-29 23:59:45 | 2013 |         333 | 2013-11-30
00:00:00 |      31018 | e84fcafb048ba6031ab0941c95908771 |
ftp://coconet1/rinex/nav/2013/333/abvi3330.13n.Z |
+-------------+-----------+----------------+---------------+--------
--+
```

To "update" (change) a field value in several rows based on another field value, in table datafile:

```
update datafile set sample_interval=15.000 where datafile_stop_time
like '%23:59:45';
Query OK, 138891 rows affected, 1 warning (2.05 sec)
```

Appendix B: Setup and Maintenance of the Downloader Software

First time setup.

Gather the following information:

1. Ops user password:
2. Database Name:
3. Database User name:
4. Database Password:
5. Host name of the system:
6. Public FTP server URL:
7. Root Data directory: (default) /data/
8. Dataworks base directory: (default) /dataworks/
9. Four character ID(s) for the station(s):
10. Station name(s):
11. Latitude, longitude and elevation of the station(s):
12. Mount type of the station(s):
13. Pick a date and time for putting the station into service:
14. Antenna type(s):
15. Antenna offset(s) in north, south, and elevation:
16. Antenna serial number(s)
17. The dome type(s):
18. Receiver model(s):
19. Receiver serial number(s):
20. Receiver firmware version(s)
21. Receiver Data sample interval(s):
22. Receiver Download intervals(s):
23. Receiver data directory path (s):
24. Receiver file name format(s):
25. Receiver file extension(s):
26. Network address of the receiver(s):
27. Organization name:
28. Organization location name:
29. Country:
30. Network name:
31. Metpack type (optional):
32. Metpack serial number (optional):

Items 1-5 are provided with the dataworks server as part of the initial setup.
Item 6 depends on your server hostname and configuration and is typically established to be ftp://<hostname>/
Items 9,10,13,27-28 and 30 are user choices determined by your organization.
Items 11,12,14-17,29, and 31-32 are determined by the physical configuration of your station and equipment.
Items 18-26 are determined by your receiver and network, section 7 figures 2a & 2b give examples of how to locate this information.

Start Here:

- Log into the server as the ops user: (see the Dataworks Administrator guide for assistance).
- Change to the /dataworks/ directory [cmd] *cd /dataworks/*

Adding a new station to the database (after completing start here):

- Change to the db_aids/ directory [cmd] *cd db_aids*
- Using a text editor edit the template file station_data.csv [cmd] *nano station_data.csv*
- In the template file enter the information from above items 9-20, and 25-30 into the comma separated file in the correct order. **NOTE** no commas (,) or apostrophes (') are allowed in any field entry.
- Once you have finished creating your station file you can save it to another name (eg TEST.csv) with the command [cmd] *[control]O STATION_NAME.csv*
- Exit the editor [cmd] *[control]X*
- Using a python script the data in the csv file is then registered into the database. In the following command example dbname is from item 2 above. Dbacctname is from item 3, dbacctpw is from item four. Section 7.1 gives a detailed example of using the script and creating the csv file. [cmd] *./insertNewStation.py STATION_NAME.csv localhost dbacctname dbacctpw dbname*
- A message 'Inserted 1 new station without problems.' indicates a success. See section 7.1 for additional help if you received an error message.

Adding a new receiver configuration to the data downloader (after completing start here):

- Change to the lib/ directory [cmd] *cd lib*
- List the existing types of receivers [cmd] *ls conf*
- Use the python script to add a new receiver configuration [cmd] *./changeReceiverType.py*
- Enter the answer to the prompts. Receiver model is item 18. Receiver Configuration sub group should be different than any thing that showed up when you typed (ls conf). It is unique to this single receiver session setup. Datapath is item 23, interval is item 22, filename format is item 24, file extension is item 25, comments are user text to help identify the configuration choices.
- Optional if set up for graphical interface (see section 7.2) you can use the optional command [cmd] *./changeReceiverType.py -vis*
- Optional if you have files in the conf/ directory you want to modify use the command [cmd] *./changeReceiverType.py -update NetR9.1* [or cmd] *./changeReceiverType.py -update NetR9.1 -vis*

Adding a new station configuration to the data downloader (after completing start here):

- Change to the lib/ directory [cmd] *cd lib*
- List the existing receiver types [cmd] *ls conf*
- List the existing stations [cmd] *ls ../stations*
- Use the python script to add a new receiver configuration [cmd] *./changeStation.py*
- Enter the answer to the prompts. Station ID depends on your receiver setup (section 7 figures 2a & 2b). If the files begin with the station serial number use item 19, otherwise you may use item 9 from above the four char id. Receiver model is item 18, receiver type is selected from the list provided (ls conf). Connection type, select from the list provided. Connection resource is item 26, Number of download days is how far back you want to check for new files on the receiver, Importance is the priority of download, user and T password are for receivers requiring authentication. The alias is only required if you used a serial number (item 19) as your station id. If you did then enter item 9 in this block. The last block is user comments.
- Optional if set up for graphical interface (see section 7.2) you can use the optional command [cmd] *./changeStation.py -vis*
- Optional if you have files in the conf/ directory you want to modify use the command [cmd] *./changeStation.py -update IRID* [or cmd] *./changeStation.py -update IRID -vis*

Configuring your download manager for startup (after completing start here):

- Change to the ops/ directory [cmd] *cd ops*
- Edit the configuration file [cmd] *nano downloadManager.conf*
- The file has the following content : *{"compact": true, "dlPause": 60, "maxProcess": 5, "terminate": false, "aliveReport": 20, "dlType": ["IP", "URL"]}* (section 7.2 on the downloadManager describes the intent of the variables) For a less active manager increase dlPause. To download more stations at once increase maxProcess. Default values are meant as a functional start point.
- Exit the editor and save changes [cmd] *[control]X*

Configuring your download exporter for startup (after completing start here):

- Change to the ops/ directory [cmd] *cd ops*
- Edit the configuration file [cmd] *nano downloadExporter.conf*
- The file has the following content : *{"xpPause": 60, "maxProcess": 10, "terminate": false, "aliveReport": 20, "ftpRoot": "/data/", "ftpPrefix": "ftp://myhost.eu/"}* (section 7.2 on the downloadExporter describes the intent of the variables) For a less active exporter increase xpPause. To process more data files at one time increase maxProcess. FtpRoot is item 7, and ftpPrefix is item 6 from above. Default values are meant as a functional start point with the exception of ftpPrefix which must be changed to correctly export data to the dataworks database.
- Exit the editor and save changes [cmd] *[control]X*

Configuring your download ingester for startup (after completing start here):

- Change to the ops/ directory [cmd] *cd ops*
- Edit the configuration file [cmd] *nano downloadIngester.conf*
- The file has the following content : *{"maxProcess": 10, "terminate": false, "ingPause": 60, "aliveReport": 20}* (section 7.2 on the downloadIngester describes the intent of the variables) For a less active ingester increase ingPause. To increase the number of db connections to register data faster increase maxProcess. Default values are meant as a functional start point.

- Exit the editor and save changes [cmd] *[control]X*

- Configuring downloader components to access the database (after completing start here):
- Change to the lib/conf/ directory [cmd] *cd lib/conf*
- Edit the configuration file [cmd] *nano dbaccessor.conf*
- The file has the following content : *{"user": "dataworks", "password": "passtodbaccess", "host": "127.0.0.1", "database": "dataworks"}* User is set from item 3 above. Password is item 4, and database is set from item 2. Host points to the internal address of the local machine and should not need to be changed.
- Exit the editor and save changes [cmd] *[control]X*
-

- Configuring downloader components to access the database (after completing start here):
- Change to the lib/conf/ directory [cmd] *cd lib/conf*
- Edit the configuration file [cmd] *nano dbaccessor.conf*
- The file has the following content : *{"user": "dataworks", "password": "passtodbaccess", "host": "127.0.0.1", "database": "dataworks"}* User is set from item 3 above. Password is item 4, and database is set from item 2. Host points to the internal address of the local machine and should not need to be changed.
- Exit the editor and save changes [cmd] *[control]X*

Starting the download manager (must have configurations completed)

- Change to the ops/ directory [cmd] *cd /dataworks/ops/*
- Use the manager start script to enable the service [cmd] *./dlManager_start &*
- Check the log file for activity [cmd] *tail -f ../logs/downloadManager.py.log*
- Look for the presence of wget entries indicating downloads
- Exit the log file [cmd] *[control] C*
- See the troubleshooting section for common problems

Starting the download exporter (must have configurations completed)

- Change to the ops/ directory [cmd] *cd /dataworks/ops/*
- Use the exporter start script to enable the service [cmd] *./dlExporter_start &*
- Check the log file for activity [cmd] *tail -f ../logs/downloadExporter.py.log*
- Look in the log file for the mention of diary creations.
- Exit the log file [cmd] *[control] C*
- See the troubleshooting section for common problems

Starting the download ingester (must have configurations completed)

- Change to the ops/ directory [cmd] *cd /dataworks/ops/*
- Use the ingester start script to enable the service [cmd] *./dlIngester_start &*
- Check the log file for activity [cmd] *tail -f ../logs/downloadIngester.py.log*
- Look in the log file for successful registry of data files in the database
- Exit the log file [cmd] *[control] C*
- See the troubleshooting section for common problems

Shutting down the downloader components:

- Change to the ops/ directory [cmd] *cd /dataworks/ops/*
- Use the component stop scripts to disable the service [cmd] *./dlManager_stop* [or cmd] *./dlExporter_stop* [or cmd] *./dlIngester_stop*
- Check the log files to confirm services have stopped (see previous sections)
- Individual components may be shut down without adversely effecting the other services. You may wish to shut down the download ingester for database maintenance periods.

Steps to change equipment session:

- Making use of the previously created csv file to enter a new station into the dataworks database, users can modify the content of the file and run the insert script a second time to manage equipment session changes.

Troubleshooting common downloader problems:

- downloadManager won't retrieve files:
  - Use an internet connection to see if the station is reachable.
  - See the steps on adding a new station. Use the -update option on the station to verify the correct settings for network address.
  - Change to the station data directory and look for the presence of a .wget.log file. The contents of the file will help you determine if the attempt to download data from the station is encountering problems.
  - See the steps on adding a new receiver. Use the -update option on the station to verify the correct settings for the receiver if the wget.log indicates problems with file names or file paths.
  - Check the /dataworks/.hold/ directory. When a station ID or an alias isn't found in the database the download files are moved to the hold directory by the exporter.
  - Check the lib/conf/dbaccessor.conf file to ensure the downloader can access the dataworks db to register the files for use in gsac.
  - Restart the downloadManager by the start script.
  - Contact the dataworks email group for further assistance.
- Can't retrieve files from the GSAC service:
  - Check to see that the station ID is registered in the dataworks database by searching GSAC for the four char ID used in the station configuration file.
  - Check to see that the exporter and ingester files are running
  - Check to see that the ftpPrefix is set correctly in the downloadExporter.conf file. If you make changes then restart the exporter.
  - Contact the dataworks email group for further assistance.
