**Global Precipitation Mission (GPM)**

**Ground Validation System**

**GPM Validation Network Data Acquisition and Preprocessing Software**

**August 26, 2016**

Goddard Space Flight Center

Greenbelt, Maryland 20771



**Document History**

| Document Version | Date | Changes |
| --- | --- | --- |
| Draft | February 8, 2010 | Initial draft |
| 2nd Draft | November 18, 2010 | Updated to reflect new configuration after move of ingest and primary data storage and archive from ws1-gpmgv to ds1-gpmgv. |
| 3rd Draft | March 28, 2012 | Update figures and URLs.  Updates for Version 7 operational status.  Update V7 CT file naming convention.  Update database table definitions.  Update ds1-gpmgv backup procedure descriptions. |
| 1.0 | June 25, 2012 | - Define PPS, TMI, and GMI acronyms.  - Added NAM model grid data acquisition for sounding data, Sections 2.5 and 4.7.  - Added GR metadata processing description, Section 4.7.1.  - Updated Figs. 1 and 9, add Figs. 10 and 11.  - Updated script initiation methods to represent current crontab list for user ‘gvoper’ on ds1-gpmgv, as all scripts are now scheduled to run automatically via cron.  - Add new database table definitions supporting NAM grid acquisition. |
| 1.1 | September 18, 2012 | Updated URL for NWS RIDGE radar mosaic downloads, and MOSAIC\_DATA environment variable of download script. |
| 1.2 | September 4, 2013 | Updated catalogXXXradar.sh sections’ schematic figures and listings files naming conventions. |
| 2.0 | December 31, 2014 | First GPM-era version, major rewrite.  Removed catalogRAWradar.sh section. |
| 2.1 | August 26, 2016 | - Replaced NOAA/NCEP grid file processing with NAM model sounding file processing description.  - Updated orbit subset definitions table.  - Added 1C-R-XCAL to the list of GPM and Constellation products ingested.  - Added GPM ground track (GT-7) prediction file and GPM orbit definition (ORBDEF.txt) file acquisition and processing descriptions.  - Added descrition and definition of ‘gpm\_orbits’ database table.  - Added subsection on PPS ITE data acquisition.  - Added section on Status Reporting and Data Archiving scripts.  - Added definitions of the three new Australian orbit subsets. |

**Contact Information**

Additional information, including information on VN points-of-contact, can be obtained from the GPM Ground Validation web site:

http://pmm.nasa.gov/science/ground-validation

Table of Contents

1. Introduction 1

2. Data Types, Sources, and Acquisition Methods 2

2.1 GPM Core and Constellation Satellite data 2

2.1.1 PPS Internal Test and Evaluation (ITE) data products 4

2.2 Ground Radar (GR) data 5

2.2.1 GR data directory structure 7

2.3 Coincidence Table (CT) files 7

2.3.1 Special Site Coincidence Situations 8

2.3.2 CT data file directory structure 8

2.4 NWS RIDGE National Radar Mosaics 8

2.5 NOAA North American Mesoscale (NAM) Model Soundings 9

2.6 GPM 7-day Ground Track files 10

2.7 GPM Orbit Definition Files 10

3. Hardware and Software Environment 12

3.1 Script locations 12

4. Data Acquisition Scripts 13

4.1 get\_PPS\_CS\_data.sh (getPPSftpListings.sh, get\_PR\_DPR\_Meta.sh) 14

4.1.1 Logic outline of get\_PPS\_CS\_data.sh 17

4.1.2 Database usage 19

4.2 get\_PR\_DPR\_Meta.sh 20

4.2.1 get2A23-25Meta.sh (get2A23Meta.sh, get2A25Meta.sh, loadmetadata.sh) 21

4.2.1.1 Database usage 22

4.3 getDPRMeta.sh (get2ADPRMeta4date.sh, loadmetadata.sh) 22

4.4 wgetCTdailies.sh (RidgeMosaicCTMatch.sh) 23

4.4.1 Logic outline of wgetCTdailies.sh 24

4.4.2 Logic outline of RidgeMosaicCTMatch.sh 26

4.4.3 Database usage 28

4.5 wgetRidgeMosaic-latest.sh 29

4.6 catalogQCradar.sh 30

4.6.1 Database usage 32

4.7 wget\_GT7\_GPM.sh (extract\_daily\_predicts.sh) 32

4.7.1 Logic outline of wget\_GT7\_GPM.sh 33

4.7.2 Logic outline of extract\_daily\_predicts.sh 34

4.7.3 Database usage 35

4.8 wget\_orbdef\_GPM.sh (extract\_orbits) 38

4.8.1 Logic outline of wget\_orbdef\_GPM.sh 38

4.8.2 Logic outline of extract\_orbits() 39

4.8.3 Database usage 39

5. The ‘gpmgv’ database in PostgreSQL 42

5.1 Dynamic tables 42

5.2 Static Tables 48

6. Status Reporting and Data Archiving scripts 52

7. Backup scripts and procedures 53

# Introduction

The Global Precipitation Measurement (GPM) Validation Network (VN) compares data from the satellite-borne TRMM Precipitation Radar (PR), GPM Dual-frequency Precipitation Radar (DPR) and GPM, TRMM and constellation satellite Microwave Imagers (GMI, TMI, etc.) to ground-based scanning weather radar (GR) observations, to assess the first-order differences between the satellite- and ground-based instruments and identify conditions under which there are discrepancies between their observations of reflectivity and rain rate. The TRMM PR and TMI were used as surrogates for the GPM Dual-frequency Precipitation Radar (DPR) and GPM Microwave Imager (GMI) for development of VN procedures prior to GPM’s launch. This document describes the suite of software used by GPM Ground Validation (GPM GV) to acquire, catalog, store, and preprocess GPM core and constellation satellite data and coincident GR data in support of the VN. Separate documents describe the software that performs the volume matching of PR, DPR, and microwave imager data with GR data; and software to analyze, compare, and display the various types of volume-matched data. The VN data acquisition and preprocessing software is portable and adaptable, but not generic. It has been written to run in a specific environment (as bash shell scripts in Linux), and requires specific COTS and freeware packages (IDL and the PostGRESQL RDBMS). These coding and environment details will be described in Sections 3 and 4.

# Data Types, Sources, and Acquisition Methods

The VN acquires and processes seven types of data on a regular or ad-hoc basis:

1. TRMM PR products 1C-21, 2A-23, 2A-25, and PR/TMI combined 2B-31 (inactive following the de-orbit of the TRMM satellite)
2. GPM DPR products 2A-DPR, 2A-Ka, and 2A-Ku, and DPR/GMI combined 2B‑DPRGMI
3. Microwave imager rain products 1C-R-XCAL and 2A-GPROF for GPM and constellation satellites (including TRMM)
4. GR volume scans and/or RHI scans, usually in Universal Format (UF) data files
5. daily satellite-GR Coincidence Table (CT) files;
6. atmospheric sounding data coincident in time and space with the GR and TRMM or GPM data, in the form of model derived soundings (ad-hoc acquisition);
7. GPM 7-day Ground Track Prediction (GT-7) files, produced daily; and
8. GPM orbit definition files, one per orbit.

Though not required, the VN also routinely acquires GIF image files of the NWS National Radar Mosaic (RIDGE mosaics). Each of these data types and a high-level overview of their methods of acquisition are described below. Detail on the acquisition scripts is provided in Section 4.

## GPM Core and Constellation Satellite data

Six types of GPM DPR, GMI, and Combined DPR/GMI Level 1 and 2 products are acquired in the VN system: 2A-DPR, 2A-Ka, and 2A-Ku from the DPR; 1C-R-XCAL and 2A-GPROF from the GMI; and 2B-DPRGMI from the combined DPR/GMI rain algorithm. For constellation satellites excluding TRMM, only the 1C-R-XCAL and 2A-GPROF data products from the scanning microwave imagers is acquired. These satellites and their microwave imagers include: F16/SSMIS, F17/SSMIS, F18/SSMIS, GCOMW1/AMSR2, METOPA/MHS, METOPB/MHS, NOAA18/MHS, and NOAA19/MHS.

Five types of TRMM PR and TMI Level 1 and Level 2 products were acquired in the VN prototype (i.e., prior to the GPM launch): 1C-21 (PR Reflectivities – raw calibrated); 2A-12 (TMI Profiling); 2A-23 (PR Qualitative – rain/no rain flag, bright band analysis, rain type); 2A-25 (PR Profile – attenuation-corrected reflectivity, 3-D rain rate, rain flag, range bin numbers, near-surface rain), and 2B-31 (TRMM Combined – combined PR/TMI 3-D and near-surface rain rates). These PR products continued to be acquired in the GPM era prior to the shutoff of the TRMM instrument systems, in addition to the 2A-GPROF product from the TRMM TMI.

In the normal operating mode, data files for these products are acquired via password-protected ***ftp*** from the NASA Precipitation Processing Subsystem (PPS). GPM GV has established “subscriptions” for these data products with the PPS for a specific set of “orbit subsets” (fixed geographical areas). The PPS generates these orbit subset products above whenever a satellite’s orbit results in the observing instrument’s footprints overlapping the geographic area defined for the subset. The resulting product file names contain the subset identifier assigned to the given orbit subset area (e.g., 1C21.090605.65834.6.sub-GPMGV1.hdf.gz for the pre-GPM “sub-GPMGV1” subset ID).

The primary orbit subset used in the pre-GPM VN was the “sub-GPMGV1” subset covering 21 WSR-88D sites in the southeastern U.S., and is defined by rectangular latitude/longitude boundaries extending from 24° to 35° N latitude, and -80° to -98° E longitude. The other two orbit subsets routinely acquired by the VN were the “GPM\_KMA” subset covering the South Korean weather radars, and the “KWAJ” subset covering only the single radar on Kwajalein atoll. The GPM\_KMA subset for the VN is defined by rectangular latitude/longitude boundaries extending from 31° to 36° N latitude, and 123.7° to 131.5° E longitude. The pre-GPM KWAJ subset is a legacy subset processed by the Goddard Earth Sciences Data and Information Services Center (GES DISC). KWAJ subset data were obtained from the DISC public ***ftp*** site rather than from a PPS ***ftp*** site. The TMI 2A-12 product was not acquired for the GPM\_KMA and KWAJ orbit subsets.

Orbit subsets for the GPM area, current as of this writing, are listed below in Table 1. All products for GPM and the constellation satellites are produced for all subsets except DARW, and the AKradars subset is not applicable to TRMM due to its low inclination orbit path. DARW subsets only apply to GPM and TRMM. The CONUS and KOREA subsets are geographically-expanded GPM-era replacements for the sub-GPMGV1 and GPM\_KMA subsets of the pre-GPM era. The subsets BrazilRadars, Guam, Hawaii, and SanJuanPR were defined after the TRMM re-entry, so no TRMM data exist for TRMM versions 6 or 7.

**Table 1.** Coincidence Subset geographical definitions for VN orbit subsets.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CSI Name** | **Description** | **North latitude bound** | **South latitude bound** | **West longitude bound** | **East longitude bound** |
| AKradars | Middleton Island and Nome WSR-88D radars, Alaska | 66.5 N | 55.0 N | -167.0 E | -134.0 E |
| AUS-East | Eastern half of Australian continent | -9.4 N | -40.14 N | 128.67 E | 155.49 E |
| AUS-West | Western half of Australian continent | -15.7 N | -37.19 N | 111.42 E | 130.55 E |
| BrazilRadars | Brazilian radars | -3.0 N | -23.0 N | -57.0 E | -32.0 E |
| CONUS | Contiguous 48 United States | 50.0 N | 23.0 N | -126.0 E | -66.0 E |
| DARW | Darwin, Australia CPOL radar | -14.5 N | -10.0 N | 128.74 E | 133.35 E |
| Guam | Guam WSR-88D radar | 15.7 N | 11.2 N | 142.56 E | 147.06 E |
| Hawaii | Hawaii WSR-88D radars | 24.15 N | 16.85 N | -162.2 E | -153.32 E |
| KOREA | Korean radars | 32.5 N | 39.0 N | 124.5 E | 130.5 E |
| KWAJ | Kwajalein KPOL radar | 6.47 N | 10.97 N | 165.47 E | 170.01 E |
| SanJuanPR | San Juan, Puerto Rico, WSR-88D | 20.37 N | 15.87 N | -68.35 E | -63.85 E |
| Tasmania | Radars on Tasmania, Australia | -38.93 N | -45.36 N | 143.33 E | 150.06 E |

Acquisition of the orbit subsets from PPS is handled by the ***bash*** script get\_PPS\_CS\_data.sh (Section 4.1), which is run once daily under the crontab. The script get\_PPS\_CS\_data.sh accesses the PPS to download any new VN-specific orbit subset products.

If get\_PPS\_CS\_data.sh is successful in downloading any new PR or DPR products, then it will also call a child script get\_PR\_DPR\_Meta.sh, which is a wrapper to run the two rain metadata extraction scripts and get2A23-25Meta.sh and getDPRMeta.sh. The script get2A23-25Meta.sh runs a set of IDL procedures to extract information about the state of the GR site overpass event(s) for the given TRMM orbit in terms of presence of precipitation echoes in the PR data and the amount of overlap between the PR data swath and the GR scan area. These metadata are written to the VN database to facilitate later processing and identification of significant precipitation events for each GR site. Likewise, getDPRMeta.sh runs an IDL procedure to extract precipitation and overlap metadata between the DPR Ku-band radar data and the ground radars for GPM overpass events, and stores these metadata to the VN database.

### PPS Internal Test and Evaluation (ITE) data products

Prior to public release of a new version of GPM and constellation data products, PPS typically produces one or more test product versions (ITExxx, where xxx is the test version number) of the data products using the updated algorithms. These data products are made available to the VN for internal evaluation, and may involve some or all of the products and subsets described in 2.1, covering a range of dates decided on by the PPS and developers. A set of manually-run scripts that mirror the functionality of get\_PPS\_CS\_data.sh, update\_CS\_dirs.sh, and getPPSftpListings.sh (Section 4.1) may be run on an ad-hoc basis to ingest the data into a private storage area on the VN data server once a new series of ITE data have completed. The current versions of these ad-hoc scripts are named get\_PPS\_CS\_v4testdata.sh, getPPSftpListings\_v4test.sh, and update\_CS\_dirs\_v4test.sh, and they are located on ds1-gpmgv.gsfc.nasa.gov in the /home/gvoper/scripts directory where the operational scripts also reside. The main difference between the ad-hoc and operational scripts is that the ad-hoc ITE scripts do not call child scripts to extract DPR metadata.

## Ground Radar (GR) data

The primary GR data used in the VN prototype was from 21 WSR-88D radars contained within the area of the sub-GPMGV1 PR orbit subset area (Table 2), and the KWAJ radar in the KWAJ subset. In the current VN baseline, 75 WSR-88D sites and the KWAJ and NPOL radars are included in GPM Validation Network (Fig. 1). These sites are defined and listed in the *GPM Validation Network Data User’s Guide, Vol. 2, GPM Data Products*.

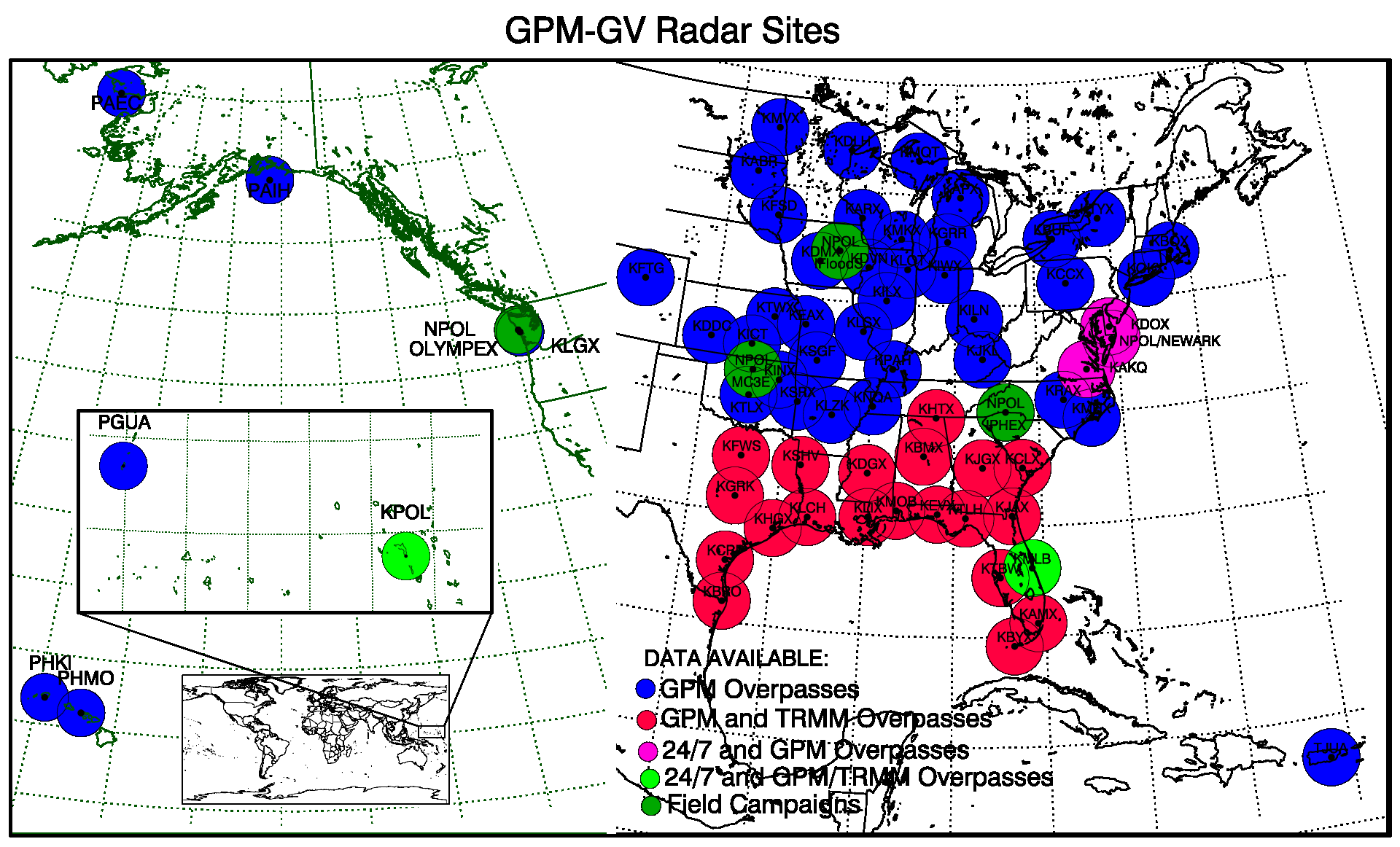
****

Figure 1. Location of primary GPM VN ground radar sites. For each site, ground radar observation limits are also illustrated.

The GPM VN system does not acquire any GR data directly from data providers on a routine basis. In this regard, the VN data acquisition software suite is a necessary but insufficient capability in its existing implementation. As most agencies will already have GR data acquisition procedures and/or a GR data archive, this is not a critical issue for duplication of the VN data acquisition procedures.

GR data for the 75 WSR-88D sites, Kwajalein (KWAJ), and NPOL are routinely acquired by the PMM Ground Validation (PMM GV) Office in support of GPM GV. The WSR-88D data are acquired by as raw NEXRAD Level II archive files, quality controlled, and processed to derive dual-polarization parameters. The original and processed GR data products are pushed to the GPM VN host machine on an ad-hoc schedule. The products include a 1CUF file in UF format that contains the quality-controlled and derived dual-pol parameters, a set of PPIs as image files, and the original, raw NEXRAD Level-II Archive files. In its current form, the VN prototype only uses the 1CUF files that have undergone the detailed quality control and include the derived parameters, but all the GR file types are permanently retained in the VN data file system.

PMM GV pushes all data files into the directory /data/gpmgv/gv\_radar/finalQC\_in on the GPM VN data server. Beneath this top-level directory are subdirectories for site ID, product type, and date. These directory structures under the common gv\_radar directory are documented in the Section 4 of the GPM Validation Network Data User’s Guide. PMM GV pushes data to the GPM VN at their convenience. Within the VN, the script catalogQCradar.sh runs once a day under the crontab to identify and catalog all new files pushed by PMM GV to the VN data server. Thus, ingest and cataloging of GR data routinely used within the VN is automated, though not completely self-contained in the VN software suite.

GR data files in Universal Format for the other ground radar sites that have contributed data to the VN have been manually acquired on an ad-hoc basis, according to the manner in which they were made available to the VN. These sites include the University of Alabama-Huntsville ‘ARMOR’ C-POL radar (RMOR), the Gosan, Korea S-band radar (RGSN), The Darwin, Australia C-POL radar (DARW), and nine Brazilian radars. The UF files for these sites have been placed in the 1CUF directory under /data/gpmgv/gv\_radar/finalQC\_in/XXXX, where XXXX is the site ID of the radar, either by hand or by running a script written for this purpose (e.g., RGSN\_to\_datadirs.sh). After being moved into the correct directory locations, the data in these site-specific directories is cataloged to the database by manually running one of the scripts written/adapted for this purpose (e.g., catalogQCradarDARW.sh).

Table 2. WSR-88D (plain text) and other (italic text) ground radar sites used in the pre-GPM Validation Network prototype.

| Site ID | Site Full Name | Latitude | Longitude |
| --- | --- | --- | --- |
| KAMX | Miami, FL | 25.6111 N | 80.4128 W |
| KBMX | Birmingham, AL | 33.1722 N | 86.7697 W |
| KBRO | Brownsville, TX | 25.9161 N | 97.4189 W |
| KBYX | Key West, FL | 24.5975 N | 81.7031 W |
| KCLX | Charleston, SC | 32.6556 N | 81.0422 W |
| KCRP | Corpus Christi, TX | 27.7842 N | 97.5111 W |
| KDGX | Jackson, MS | 32.3178 N | 89.9842 W |
| KEVX | Red Bay/Eglin AFB, FL | 30.5644 N | 85.9214 W |
| KFWS | Dallas-Ft Worth, TX | 32.5731 N | 97.3031 W |
| KGRK | Central Texas (Ft Hood), TX | 30.7219 N | 97.3831 W |
| KHGX | Houston/Galveston, TX | 29.4719 N | 95.0792 W |
| KHTX | N.E./Hytop, AL | 34.9306 N | 86.0833 W |
| KJAX | Jacksonville, FL | 30.4847 N | 81.7019 W |
| KJGX | Robins AFB, GA | 32.6753 N | 83.3511 W |
| KLCH | Lake Charles, LA | 30.1253 N | 93.2158 W |
| KLIX | Slidell AP/New Orleans, LA | 30.3367 N | 89.8256 W |
| KMLB | Melbourne, Florida | 28.1133 N | 80.6542 W |
| KMOB | Mobile, AL | 30.6794 N | 88.2397 W |
| KSHV | Shreveport, LA | 32.4508 N | 93.8414 W |
| KTBW | Ruskin/Tampa Bay, FL | 27.7056 N | 82.4017 W |
| KTLH | Tallahassee, FL | 30.3975 N | 84.3289 W |
| *DARW* | *Darwin, Australia* | *12.2522 S* | *131.0430 E* |
| *KWAJ* | *Kwajalein atoll, Marshall Islands* | *8.71796 N* | *167.733 E* |
| *RGSN* | *Gosan, South Korea* | *33.2942 N* | *126.1630 E* |
| *RMOR* | *University of Alabama, Huntsville* | *34.6460 N* | *86.7713 W* |

### GR data directory structure

The top-level data VN data directory (/data/gpmgv), the top-level GR product subdirectory (gv\_radar/), and the QC-level-specific subdirectory (finalQC\_in/) are locally configurable locations. Editing of baseline scripts is required to change these values. The lower-level subdirectory beneath the QC-level directories is not locally configurable (without massive modification of code and scripts) and must be present as defined in the VN baseline for the VN data acquisition and processing script to work properly. The complete GR data directory structure is documented in Section 4 of the *GPM Validation Network Data User’s Guide, Vol. 2, GPM Data Products*.

## Coincidence Table (CT) files

The PPS produces a separate daily Coincidence Table (CT) files for GPM and each constellation satellite, which list the orbit number and time and distance of that satellite’s closest approach to each of the ground sites previously configured in the PPS for this purpose. Whenever the satellite’s orbital subtrack passes within 750 km of one of the configured ground radar sites an entry for the site overpass event is made in the daily CT. New sites are added to the CT configuration by arrangement with the PPS. All VN sites displayed in Figure 2.2-1, as well as DARW, King City (Canada), and selected KMA (Korean) radars are configured in the current CT site lists in the PPS.

The daily CT files are acquired from the PPS ftp site by the wgetCTdailies.sh script, which is run once daily via crontab. The CT filename convention is **CT.SAT.yyyymmdd.jjj.txt**, where **SAT** is the ID of the satellite, **yyyymmdd** is the year, month, and day, and **jjj** is the day-of-year of the site overpass events tabulated in the file. In normal operations, the PPS produces the CT file one more more days after the fact depending on the satellite, so wgetCTdailies.sh attempts to acquire the CT file for prior days when run.

If wgetCTdailies.sh is successful at acquiring one or more CT files, it then calls the child scripts newCT\_to\_DB.sh and RidgeMosaicCTMatch.sh to process the CT file(s). First, newCT\_to\_DB.sh identifies CT entries for stations in the VN, and reformats the resulting subset of CT data into a delimited, ASCII text file in a format suitable for loading into a PostGRESQL data table. The reformatted and subsetted data are written to a file named **CT.SAT.yyyymmdd.jjj.unl**, where **SAT.yyyymmdd.jjj** is the same value as the PPS CT file being processed, and are loaded into the ct\_temp table in the database to await additional processing. At a later time, RidgeMosaicCTMatch.sh reads the entries in the ct\_temp table, finds events where the closest approach is within 250 km of the sites, identifies overpass events that are ‘new’ to the VN database, creates entries for any new overpass events in the overpass\_event permanent table, and if successful, removes the temporarily-held overpass event data from the ct\_temp table. RidgeMosaicCTMatch.sh also performs other functions related to the processing of NWS “RIDGE” national radar mosaic images acquired by the VN (see Section 2.4). ***The events defined in the* overpass\_event *table in the VN database form the basis for all GR-site-specific event data processing in the VN***, at least as performed by the existing VN data processing scripts and supporting SQL queries.

### Special Site Coincidence Situations

It is sometimes necessary to acquire historical site overpass event information for a site not previously defined in the CT configuration, such as when adding existing data for a new GR site to the VN dataset. The web site:

http:// <http://cloudsgate2.larc.nasa.gov/cgi-bin/predict/predict.cgi>

contains a tool called the NASA LaRC Overpass Predictor that can compute satellite overpass event data for a location defined by an input latitude and longitude. The resulting HTML table can be saved to a file and processed into a file format able to loaded to the overpass\_event table, thus providing CT data for any random site, for any range of datetimes. The procedures and supporting SQL commands for doing this are documented in the file addOverpassEventsFromDelimFile.sql. For GPM only, Sections 2.6 and 2.7 describe a new VN-specific option for computing CT data for a random site and time period from GPM orbital track prediction data.

### CT data file directory structure

CT files are stored in the VN file system under the following directory structure:

/data/gpmgv

/coincidence\_tables

/YYYY

/MM

/DD

where YYYY, MM, and DD are the year, month, and day of the CT file’s datestamped file name, and of the overpass predictions contained in the file, and mirror the directory structure of the PPS ftp site. The top-level data VN data directory (/data/gpmgv) and the CT subdirectory (coincidence\_tables) are configurable locations. Editing of baseline scripts is required to change these values.

## NWS RIDGE National Radar Mosaics

The National Weather Service produces a series of radar reflectivity mosaic GIF images every 10 minutes, and posts these on public web site for viewing and/or downloading. The VN acquires all the national-scale mosaic images via the wgetRidgeMosaic-latest.sh script, which is run every 6 minutes under the crontab. RIDGE mosaics are produced in near-real-time, and only the most recent image file (named latest.gif) is retained on the web site, so the VN script runs more frequently than the mosaics are produced to make sure that all new mosaics are acquired. The newly acquired mosaics are renamed to a timestamped file name, stored in a “holding” directory, and cataloged in the VN database in the heldmosaic table to await postprocessing.

The RidgeMosaicCTMatch.sh script is launched from within the wgetCTdailies.sh script, and postprocesses the held/renamed RIDGE mosaics for the previous day(s) once the CT file(s) for the day(s) has/have been acquired. The script determines which radar mosaic times correspond to the time of TRMM and GPM site overpasses for the 70 WSR-88D sites in the continental U.S. by comparing the data times of the mosaic images to the times of the overpass events for the day. Those mosaics that match the times of the site overpasses (within a time threshold) are moved from the holding directory to archivedmosaic directory, and cataloged in the coincident\_mosaic database table. The other non-coincident mosaic data files are deleted and their entries are deleted from the heldmosaic table. Most of the computations needed to do the time matchups are performed by a series of SQL commands contained in the file new\_mosaicCTmatch.sql, which are executed by calling the *psql* utility of PostGRESQL from within the RidgeMosaicCTMatch.sh script. As part of the same data processing, the SQL commands executed from within the script handle the final steps of preprocessing of CT site overpass event data held in the ct\_temp database table, as described in the previous section.

## NOAA North American Mesoscale (NAM) Model Soundings

External to the VN data acquisition, but as part of the procedures to support the quality control and parameter derivation that is performed on raw GR data prior to the GR data being pushed to the VN file system (see Section 2.2, above), model forecast grids from the North American Mesoscale (NAM) model are acquired for domestic GR site GPM overpass event times, and soundings are extracted from these forecasts. These sounding files are produced and archived on the host hector.gsfc.nasa.gov, and are stored on that server in the directory **/gvraid/trmmgv/Soundings/RUC\_Soundings**.

As an ad-hoc VN capability, the script get\_RUC\_BB\_heights.sh, has been written to post-process these soundings files for rainy GR site overpass events for GPM overpasses and extract the 0-degrees-Celsius height (freezing level) for each such rain event and write this information to a text file. The VN analysis scripts can use these data as a substitute for the satellite-radar-detected bright band height in cases where the GPM algorithm fails to detect the bright band.

The get\_RUC\_BB\_heights.sh script is located on hector.gsfc.nasa.gov in the directory **/gvraid/ftp/gpm-validation/scripts**. The prologue of the script contains a detailed step-by-step set of instructions on how to prepare the required inputs, run the script, and copy the output back to the VN data server, ds1-gpmgv.gsfc.nasa.gov.

## GPM 7-day Ground Track files

PPS generates a 7-day prediction of GPM satellite orbit subtrack locations at 1-second intervals on a daily basis in the form of a text file. Thus, there is a 6 day overlap of the prediction data between files of adjacent dates. The daily GT-7 files are acquired from the PPS ftp site by the wget\_GT7\_GPM.sh script, which is run once daily via crontab. The GT-7 filename convention is **GT-7.SAT.yyyymmdd.jjj.txt**, where **SAT** is the ID of the satellite, **yyyymmdd** is the year, month, and day, and **jjj** is the day-of-year of the starting prediction data tabulated in the file. Only GPM predictions are acquired by the VN.

If wget\_GT7\_GPM.sh is successful at acquiring one or more GT files, it then calls the child script extract\_daily\_predicts.sh to process the GT file(s) into files that contain only a single day’s worth of 1-second subtrack predictions with no overlaps in time coverage. The extracted daily data are written to files named **GPM\_1s\_subpts.yyyymmdd.txt**, where yyyymmdd is the year, month, and day of the prediction data tabulated in the file. Neither the GT-7 nor the GPM\_1s\_subpts files are used in routine VN operations. However, they are useful in generating historical and/or future site overpass times for sites that are not configured in the PPS CT file production. The IDL procedure found in the file **/home/morris/swdev/idl/dev/common\_utils/get\_coincidence\_via\_track.pro** provides a capability to compute site overpasses from the GPM\_1s\_subpts files.

The orbit track prediction files are not considered part of the GPM Validation Network core data suite, so they are stored on the VN data server (currently ds1-gpmgv) under a local directory (/data/tmp) not visible to the public via the VN ftp server.

## GPM Orbit Definition Files

PPS generates a set of GPM satellite single-orbit definition files on an ongoing basis as the orbits occur, in the form of a text file listing the orbit number and its starting and ending date and time along with other fields. The orbit definition files are acquired from the PPS ftp site by the wget\_orbdef\_GPM.sh script, which is run once daily via crontab. The orbit definition filename convention is:

**GPMCORE.yyyymmdd.hhmmssddd\_YYYYMMDD.HHMMSSDDD.001.ORBDEF.txt**

where the elements of the file name are as described in Section 4.8. Only GPM orbit definition file are acquired by the VN. The orbit track prediction files are not considered part of the GPM Validation Network core data suite, so they are stored on the VN data server (currently ds1-gpmgv) under a local directory (/data/tmp) not visible to the public via the VN ftp server.

If wget\_orbdef\_GPM.sh is successful at acquiring one or more GT files, it then calls the internal function extract\_orbits() to extract the GPM orbit number and starting and ending date/times and format these fields for loading to PostGRESQL. The extracted orbit data are written to the ‘gpm\_orbits’ table in the ‘gpmgv’ database. Site overpasses computed by the IDL procedure **get\_coincidence\_via\_track.pro** described above do not include orbit numbers, so the data in the gpm\_orbits table can be merged with the coincidence time to determine their orbit number so the the new coincidence events can be loaded to the ‘gpmgv’ database table ‘overpass\_event’. The SQL commands contained in the file **/home/morris/swdev/scripts/load\_coincidences\_via\_track.sql** comprise the series of commands needed to merge the overpass times with the orbit numbers and load them to the operational database table ‘overpass\_event’ in the ‘gpmgv’ database. The end-to-end process to manually compute and load site overpass data for new sites to the operational database is described in the *Validation Network Maintenance Manual*.

# Hardware and Software Environment

From August 2006 until mid-2010, the VN data acquisition software was hosted on a garden-variety Intel Xeon desktop workstation running Red Hat Enterprise Linux (currently RHELv4), with a 250 gigabyte (GB) hard drive capacity. It is currently running on a Dell PowerEdge T610 Server with two Intel Xeon 5520 2.26 GHz processors, 16 GB of memory, and multiple hard drives in a RAID configuration, running the CentOS operating system. Beyond what is required to run IDL and PostgreSQL, there are no stringent processor, memory, or performance requirements for the VN data acquisition host system.

All data acquisition and cataloging scripts are written in ***bash*** (the GNU Bourne-Again SHell). Scripts interact with the PostGRESQL relational database using the ***psql*** command-line utility, using SQL commands specified in the script or contained in a separate SQL command file. The PostgreSQL database server process is named **postmaster**, and runs continuously on the VN host machine under a reserved user ID. All the scheduled data acquisition scripts require the ability to read and write to the PostgreSQL database, so the server process must be available at all times. If the **postmaster** process fails, it is restarted by one of the scheduled scripts, most likely by wgetRidgeMosaic-latest.sh script, which runs with the greatest frequency (every 6 minutes). As noted in Sections 2.1 and 4.7.1, **IDL** is needed to extract metadata from all new TRMM 2A-23 and 2A-25 PR products and 2A-53 and 2A-54 GR products and GPM 2AKu products acquired by the VN. A licensed copy of **IDL** is needed, as the **IDL** procedures being run must be able to create and write output data files.

With some effort, it would be possible to modify the VN data acquisition scripts to run under a different Linux/Unix shell than bash, and to use a different RDBMS than PostgreSQL. There is no practical substitute for **IDL**, though **IDL** is only used here for the metadata extraction from the PR and GR products, not for the basic data acquisition, storage, and cataloging of the VN data. Beyond data acquisition, **IDL** is the primary data analysis and display tool used within the VN software suite, though a fully-licensed copy is only mandatory if generating new or custom sets of geometry-matched PR and GR data, or for directing the statistical analysis output results to a Postscript file. ***It is neither practical nor desirable to try to implement the GPM VN capabilities without the use of* IDL *and the* PostgreSQL *RDBMS or its like.***

## Script locations

All the operational versions of the data ingest and processing scripts that run under the **crontab** are located within the /home/gvoper/scripts directory, and are run as user ‘*gvoper*’. IDL routines called by the metadata extraction scripts are under /home/gvoper/idl, and a listing of the *gvoper* **crontab** entries is found in the file /home/gvoper/appdata/crons.gvoper.

All special, one-time, and manually-run data ingest and processing scripts, data backup scripts, and SQL command files and collections are located under /home/morris/swdev/scripts except for get\_RUC\_BB\_heights.sh, whose location is described in Section 2.5, above.

# Data Acquisition Scripts

As already described in Section 2, routine VN data acquisition is performed by a series of ***bash*** shell scripts run on a scheduled basis via the ***unix/linux*** crontab capability. Figure 2 presents an overview of the VN data sources, scripts to acquire and/or catalog these data, and the VN relational database and file system. Major scripts are shown in gray-shaded numbered rectangles in Fig. 2, with child and supporting scripts and procedures shown in parentheses/italics below the primary script names. File stores within fixed directory structures are shown as slant parallelograms in the figures, and the ***PostgreSQL*** relational database is shown as a cylinder. External ***http*** and ***ftp*** data sites are indicated by the curved parallelograms, and the external PMM GV system is shown as a rectangle with rounded corners. These same shape conventions will apply to subsequent figures showing more detail on each of the VN data acquisition and cataloging scripts.

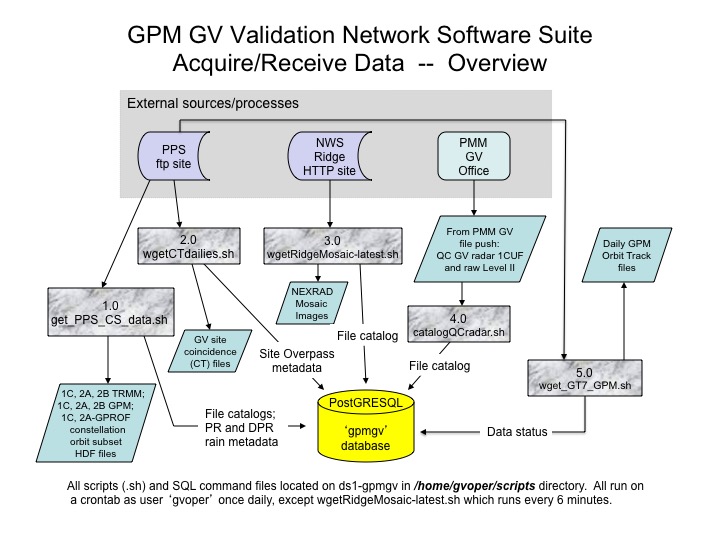


Figure 2. Overview of GPM Validation Network data sources and data acquisition scripts.

As seen in Fig. 2, there are five routinely-scheduled data acquisition/cataloging scripts:

* get\_PPS\_CS\_data.sh
* wgetCTdailies.sh
* wgetRidgeMosaic-latest.sh
* catalogQCradar.sh
* wget\_GT7\_GPM.sh

Each of these scripts and their child scripts will be described in detail in the following sections.

## get\_PPS\_CS\_data.sh (getPPSftpListings.sh, get\_PR\_DPR\_Meta.sh)

Acquisition of VN-specific orbit subset products from the PPS and DISC is handled by get\_PPS\_CS\_data.sh (Fig. 3), and calls two other scripts/utilities to do the actual downloading of new PPS product files. First, the ***bash*** child script getPPSftpListings.sh is called to manage and track the acquisition of the ftp\_url\_YYYYMMDDhhmm.txt files from the PPS, which contain listings of new PPS products created by the PPS for the VN data subscription. The Perl script mirror (Fig. 4) is then called by getPPSftpListings.sh to download the daily listings of new VN orbit subset products processed and posted for download by the PPS. As the name implies, mirror compares daily listings files present in the VN data system to those present on the external PPS ***ftp*** site, and downloads those “ftp\_url” listings files that are not in the VN dataset. Once a set of new “ftp\_url” listings files has been acquired, the paths to the products of interest are extracted from the listings files by get\_PPS\_CS\_data.sh, which then invokes the GNU wget utility to do the actual orbit subset product file transfers from the PPS ***ftp*** site. If the script logic detects that files for a new version of PPS data have been downloaded, it will call the utility script update\_CS\_dirs.sh to prepare the new baseline directory subtrees needed to store the data products in the VN data archive.

If any PR or DPR products are successfully downloaded, then get\_PPS\_CS\_data.sh also then calls get\_PR\_DPR\_Meta.sh to preprocess the PR and/or DPR data by analyzing selected PR and DPR fields to a set of temporary grids, extracting GR-site-specific information about the PR/GR and DPR/GR data overlap and precipitation coverage within the overlap area, and writing the extracted metadata to the PostGRESQL database.

A fixed configuration file for mirror is used in each run of get\_PPS\_CS\_data.sh, such that all “ftp\_url” listings files in the VN database and on the PPS sites are considered by mirror when comparing the files present in the VN data set to the files present on the PPS ***ftp*** sites. The mirror configuration file is named arthurhou.pps.eosdis.nasa.gov, which is the same as the PPS ***ftp*** site address, by design. Detailed documentation for the ***Perl*** script mirror is contained in the subdirectory under which mirror and its supporting scripts and files are located (MIR\_BIN\_DIR -- refer to Table 3). The configuration file contains instructions to download the ‘ftp\_url’ listings files from *arthurhou.pps.eosdis.nasa.gov*.

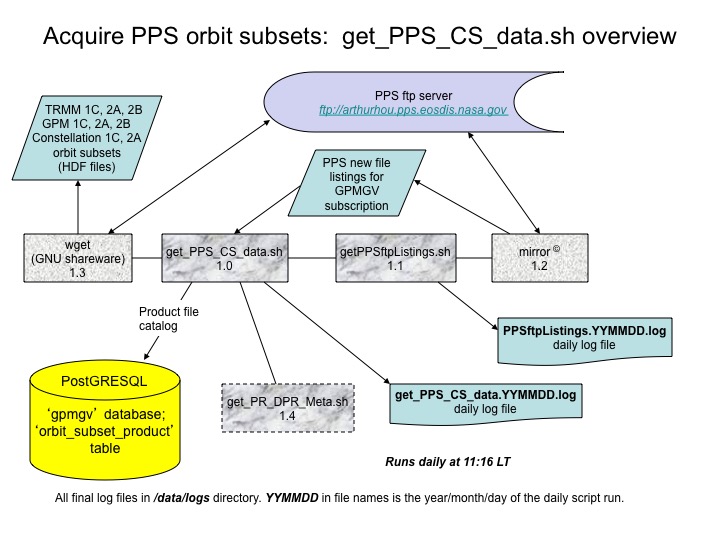


Figure 3. Overview of GPM core and constellation satellite data acquisition processes and input/output data sources.

All local configuration variables required by the ***bash*** scripts get\_PPS\_CS\_data.sh and getPPSftpListings.sh are internally defined at the top of each script. No variables are exported or passed between the two scripts. Variables for get\_PPS\_CS\_data.sh that can or must be configured locally are listed in Table 3, in the order in which they appear in the script. Table 4 lists the configuration variables used in the getPPSftpListings.sh script. In each of these scripts, the variable $rundate is the current system date in the format yymmdd, internally computed in the script at run time, and is used to make date-specific entries in the database and generate date-specific filenames.

**Table 3. Locally-configured variables defined in get\_PPS\_CS\_data.sh.**

| **Variable name** | **Baseline value** | **Description** |
| --- | --- | --- |
| GV\_BASE\_DIR | /home/gvoper | Top-level directory under which the data ingest scripts/subdirectories are located. By convention, it is also the home directory of the user who owns the crontab used to launch the scripts. In many scripts this value is internally configured based on the user ID. |
| DATA\_DIR | /data/gpmgv | Top-level directory-in-common under which all ingested/output VN data files are located. This can exist as a link to an actual directory in another location. |
| CS\_BASE | ${DATA\_DIR}/orbit\_subset | Baseline top-level directory under which downloaded orbit subset data files are written into a directory structure after being downloaded from the PPS. See prologue of the function wgetCStypes4date() inside get\_PPS\_CS\_data.sh |
| TMP\_CS\_DATA | /data/tmp/PPS\_CS | “Holding” directory to which orbit subset files are originally downloaded from the PPS. |
| BIN\_DIR | ${GV\_BASE\_DIR}/scripts | Directory where ingest scripts are located |
| LOG\_DIR | /data/logs | Directory to which scripts write their log files |
| LOG\_FILE | ${LOG\_DIR}/get\_PPS\_CS\_data.${rundate}.log | File pathname to get\_PPS\_CS\_data.sh log file. Filename is date-specific, as determined by the value of $rundate, which is computed as the current day’s date at the time of the run. |
| DBTEMPFILE | ${TMP\_CS\_DATA}/dbtempfile | File pathname to temporary file to which output from a query to the PostGRESQL database is written. |
| FILES2DO | ${TMP\_CS\_DATA}/DatesToGet | Temporary file listing partial datestamp YYMMDD of ftp\_url files to be processed this run |
| SATSUBTYPE\_TEMP | ${TMP\_CS\_DATA}/SatSubsetSubtype2do | Temporary file listing pathnames of data files to be processed for an iteration of satellite/subset/productType, used in internal ***bash*** function wgetCStypes4date() |

**Table 4. Locally-configured variables defined in getPPSftpListings.sh.**

| **Variable name** | **Baseline value** | **Description** |
| --- | --- | --- |
| GV\_BASE\_DIR | /home/gvoper | As in Table 3 |
| TMP\_DIR | /data/tmp | Location of temporary files used in the script. |
| LOG\_DIR | /data/logs | As in Table 3 |
| BIN\_DIR | ${GV\_BASE\_DIR}/scripts | As in Table 3 |
| MIR\_DATA\_DIR | ${TMP\_DIR}/PPS\_CS | Local directory to which PPS listings files are mirrored |
| MIR\_LOG\_FILE | ${MIR\_DATA\_DIR}/pps\_ftp\_lists.log | Log file to which **mirror** writes its output. Is moved/renamed to the date-specific file ${LOG\_DIR}/PPSftpListings.${rundate}.log by get\_PPS\_CS\_data.sh |
| MIR\_BIN | mirror | Name of the Perl **mirror** script itself. **Not configurable locally** |
| MIR\_BIN\_DIR | ${BIN\_DIR}/mirror | Directory under which the Perl mirror script and supporting modules are located |
| MIR\_PACKAGES\_DIR | packages | Subdirectory where mirror “package” files are located. Is relative to MIR\_BIN\_DIR, by design of mirror. **Not configurable locally** |
| MIR\_PACKAGE | ${MIR\_PACKAGES\_DIR}/arthurhou.pps.eosdis.nasa.gov | Pathname of “package” file used to configure mirror to obtain PPS listings files. |
| DBTEMPFILE | ${TMP\_DIR}/getPPSftpListings\_dbtempfile | File pathname to temporary file to which output from a query to the PostGRESQL database is written. |
| LOG\_FILE | ${LOG\_DIR}/getPPSftpListings.${rundate}.log | As in Table 3, but for getPPSftpListings.sh |

Directories and files listed in Tables 3 and 4 must already exist before get\_PPS\_CS\_data.sh and getPPSftpListings.sh can be run, with the exception of the files defined by LOG\_FILE and those described as Temporary files, which are created when the script runs. As a general rule, only the variables GV\_BASE\_DIR, TMP\_DIR, LOG\_DIR, and DATA\_DIR should need to be modified for a local installation outside of the VN baseline. The locations pointed to by these variables’ values (particularly useful for DATA\_DIR) may be set up as links to actual directories in other locations, such that the baseline value in the scripts may not need to be changed.

### Logic outline of get\_PPS\_CS\_data.sh

A high-level outline of the logic in get\_PPS\_CS\_data.sh is given below. Details are to be found in the in-line documentation in the script.

1. Define the internal variables used in the script (Table 3 and others).
2. Determine the current date from the system clock in YYMMDD format
3. Build the date-specific log file pathname and create the log file.
4. Determine whether the PostGRESQL database server is running. **If not,** send e-mail notification of the problem, and **exit the script.** Otherwise…
5. Check the appstatus table in the database (see following section) to see whether we have attempted to run get\_PPS\_CS\_data.sh for date YYMMDD, and if so, what the status of the prior run was. **If past status was SUCCESS, then note this in log file and exit.**  If no status, initialize the appstatus table to UNTRIED for this YYMMDD, write the YYMMDD value to $DBTEMPFILE, and continue. If status exists and was not SUCCESS, also continue.
6. Increment the “number of tries” for this YYMMDD in the appstatus table by 1.
7. Change working directory to TMP\_CS\_DATA so that script is pointed to location of any downloaded ftp\_url\_YYYYMMDDhhmm.txt files.
8. Execute the child script ${BIN\_DIR}/getPPSftpListings.sh to retrieve any new ftp\_url files posted to the PPS ftp server.
9. Search the log file ${LOG\_DIR}/PPSftpListings.${rundate}.log to determine whether any new ftp\_url files were downloaded. If successful (got new files), extract the ftp\_url filenames from the log file and save in the shell variable URLFILES. Otherwise, change the get\_PPS\_CS status for this YYMMDD to “MISSING” in the appstatus table and exit the script.
10. Otherwise, step through the ftp\_url files downloaded by getPPSftpListings.sh one by one, calling internal function wgetCStypes4date() to identify and download the orbit subset files of interest to the VN file system, as follows:
    1. In wgetCStypes4date(), initialize the found flag for the number of ftp\_url files found to zero.
    2. Verify that the ftp\_url file exists in the directory $TMP\_CS\_DATA. If so, then set the flag for the number of ftp\_url files found to 1.
    3. If the ftp\_url file was found, then loop over the orbit subset file names listed in the file, extract the satellite ID from the file name, and build a list of unique satellite IDs contained in the file.
    4. For each satellite ID, set up the list of orbit subset IDs and data types to be acquired for that satellite, and start a nested loop over satellite ID, subset, and data type.
    5. Build a file pattern corresponding to the satellite ID, subset, and data type, and identify any files in the ftp\_url file matching this pattern, and store the ftp\_url entries into a temp file $SATSUBTYPE\_TEMP.
    6. Call the **wget** utility with the necessary ftp parameters to download the files listed in $SATSUBTYPE\_TEMP.
    7. For each file downloaded to $TMP\_CS\_DATA, extract and/or compute the necessary metadata fields needed to determine the VN baseline directory path to the file and to catalog the file in the database.
    8. If the baseline directory path to the file does not exist, call the script update\_CS\_dirs.sh with the parameters needed to create it.
    9. If the baseline directory does not exist after the above steps, just leave the downloaded file in place in $TMP\_CS\_DATA and log the issue in $LOG\_FILE.
    10. Otherwise, move the downloaded orbit subset data file to the baseline directory and catalog the file in the orbit\_subset\_product table in the ‘gpmgv’ database (next section).
11. If wgetCStypes4date() is unsuccessful at downloading any files from the current ftp\_url file being processed, change the get\_PPS\_CS status for this YYMMDD to “$INCOMPLETE” in the appstatus table and exit the script.
12. Otherwise, change the get\_PPS\_CS status for this YYMMDD to “$SUCCESS” in the appstatus table, run the wrapper script get\_PR\_DPR\_Meta.sh, to run the IDL .bat files that execute the IDL code to extract PR and DPR metadata for the site overpass events from any downloaded TRMM PR or GPM DPR files and store the metadata in the gpmgv database table ‘event\_meta\_num'.
13. Identify any dates in the ‘appstatus’ table for the 'get\_PPS\_CS' app\_id where this is the 5th failure to attempt to download ftp\_url files. If any such dates are found, update the database to indicate “$FAILED” for these date’s run of get\_PPS\_CS\_data.sh.

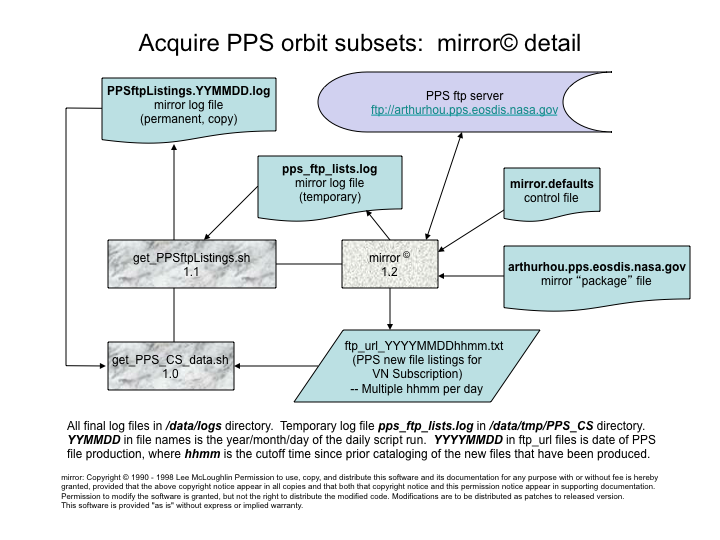


Figure 4. Detail of mirror© script usage and input/output data sources in get\_PPS\_CS\_data.sh

### Database usage

Details of the ‘gpmgv’ database design are given in Section 5. This section describes database usage pertaining to get\_PPS\_CS\_data.sh and getPPSftpListings.sh.

The get\_PPS\_CS\_data.sh script uses two ‘gpmgv’ database tables: ‘appstatus’ and ‘orbit\_subset\_product’. The appstatus table is used to track the status of the daily runs of get\_PPS\_CS\_data.sh and other scripts. Rows in the table pertaining to get\_PPS\_CS\_data.sh are identified by an ‘app\_id’ attribute value of ‘get\_PPS\_CS\_data’. Details of the various status values used in the script have been described in the logic outline, above. The script getPPSftpListings.sh uses similar logic to get\_PPS\_CS\_data.sh to download files and track its status in the ‘appstatus’ table, but under an ‘app\_id’ attribute value of ‘getPPSftpList’, using the mirror utility to do the actual new file identification and downloading.

The get\_PPS\_CS\_data script catalogs any downloaded satellite orbit subset data files in the ‘orbit\_subset\_product’ table in the ‘gpmgv’ database.

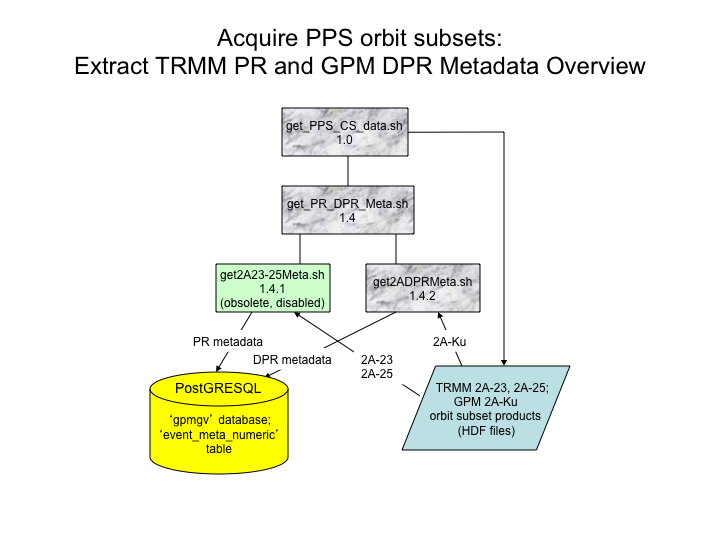


Figure 5. Overview of wrapper script get\_PR\_DPR\_Meta.sh.

## get\_PR\_DPR\_Meta.sh

If any PR or DPR products are successfully downloaded by get\_PPS\_CS\_data.sh, then get\_PR\_DPR\_Meta.sh is called. get\_PR\_DPR\_Meta.sh is a wrapper script that calls two child scripts, get2A23-25MetaNew.sh and getDPRMeta.sh (Fig. 5), after validating their existence and executable status. These two child scripts preprocess the downloaded PR (get2A23-25MetaNew.sh, *now obsolete*) and DPR (getDPRMeta.sh) data by calling IDL to analyze selected PR and DPR fields to a set of temporary grids centered over each overpassed GR site, extract information about the PR/GR and DPR/GR data overlap and precipitation coverage within the overlap area, and write the metadata to a known filename specific to the child script. The extracted metadata (if any) in the file are then written to the PostGRESQL database by the child script that produced the data. These child scripts are described in detail in the following two subsections.

### get2A23-25Meta.sh (get2A23Meta.sh, get2A25Meta.sh, loadmetadata.sh)

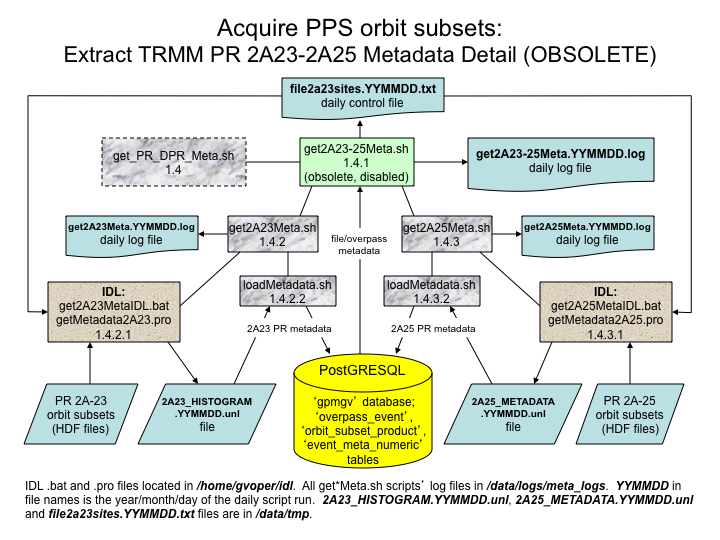


Figure 6. PR metadata extraction processes and input/output data sources for get2A23-25Meta.sh script.

If get\_PPS\_CS\_data.sh succeeds in downloading new PR data files, then get\_PPS\_CS\_data.sh calls the wrapper script get\_PR\_DPR\_Meta.sh, which in turn calls get2A23-25Meta.sh (Fig. 6) to extract metadata for each site overpass event associated with the TRMM PR orbit subset and store the metadata in the ‘gpmgv’ database.

At startup, get2A23-25Meta.sh checks its single YYMMDD date argument and the status of prior runs as tallied in the appstatus table, determines the set of dates for which metadata needs to be processed, and checks that the necessary PR and matching TRMM CT data files are available for these dates. It then prepares one or more date-specific ‘control files’ that list the 2A23 and 2A25 PR orbit subset products to be processed for the date, and the site overpass event information for the associated overpass events. The actual work of extracting and storing the metadata is done by a pair of child scripts, get2A23Meta.sh and get2A25Meta.sh, called by get2A23-25Meta.sh. In turn, these two child scripts execute IDL batch files, get2A23MetaIDL.bat and get2A25MetaIDL.bat respectively, to run IDL procedures that do the actual work of computing the metadata values for the files and events listed in the control file(s). The IDL routines store the metadata in date-specific delimited text files. Once the IDL processes complete, the respective child scripts check the status of the expected delimited text files, and if they exist and contain data, the script loadMetadata.sh is called to do the work of loading the data from the delimited text files into the ‘event\_meta\_num' table in the ‘gpmgv’ database.

*The* get2A23-25Meta.sh *script is now obsolete now that new TRMM data are no longer available. Its description is left here as a placeholder to describe the process that would be followed if new versions of historical TRMM data are to be ingested at a later date.*

#### Database usage

The scripts get2A23Meta.sh, get2A25Meta.sh, and get2A23-25Meta.sh use the ‘appstatus’ table to track the status of their daily runs, under rows identified by ‘app\_id’ attribute values of ‘get2A23Meta’, ‘get2A25Meta’, and ‘get2A2325Meta’, respectively. The script loadMetadata.sh loads data from the delimited text files into the ‘metadata\_temp’ table, checks these data rows against permanent metadata in the 'event\_meta\_num' table for duplicates, and inserts non-duplicate (new) metadata into the permanent ‘event\_meta\_num' table.

## getDPRMeta.sh (get2ADPRMeta4date.sh, loadmetadata.sh)

The getDPRMeta.sh script performs the exact same functions as get2A23-25Meta.sh, but for DPR orbit subset products and GPM site overpasses. If get\_PPS\_CS\_data.sh succeeds in downloading new DPR data files, then get\_PPS\_CS\_data.sh calls the wrapper script get\_PR\_DPR\_Meta.sh, which in turn calls getDPRMeta.sh (Fig. 7) to extract metadata for each site overpass event associated with the GPM DPR orbit and store the metadata in the ‘gpmgv’ database. At startup, getDPRMeta.sh checks its single YYMMDD date argument and the status of prior runs as tallied in the appstatus table, determines the set of dates for which metadata needs to be processed, and checks that the necessary DPR and matching GPM CT data files are available for these dates. It then prepares one or more date-specific ‘control files’ that list the DPR products to be processed and the ground site overpass event information for the associated overpass events. Of the three DPR products (2ADPR, 2AKu, 2AKa) acquired by the VN, only the 2AKu is used for metadata extraction.

The actual work of extracting and storing the metadata is done under a child script, get2ADPRMeta4date.sh, called by getDPRMeta.sh. In turn, this child script executes the IDL “batch” file get2ADPRMetaIDL.bat under IDL to set up the necessary input parameters and run the precompiled IDL procedure getmetadata2adpr.sav that does the computation of the metadata values for the files and events listed in the control file(s). The IDL routine stores the extracted metadata in date-specific delimited text files with a well-known pathname. Once the IDL process completes, get2ADPRMeta4date.sh checks the status of the expected delimited text files, and if they exist and contain data, the script loadMetadata.sh is called to load the data from the delimited text files into the ‘event\_meta\_num' table in the ‘gpmgv’ database.

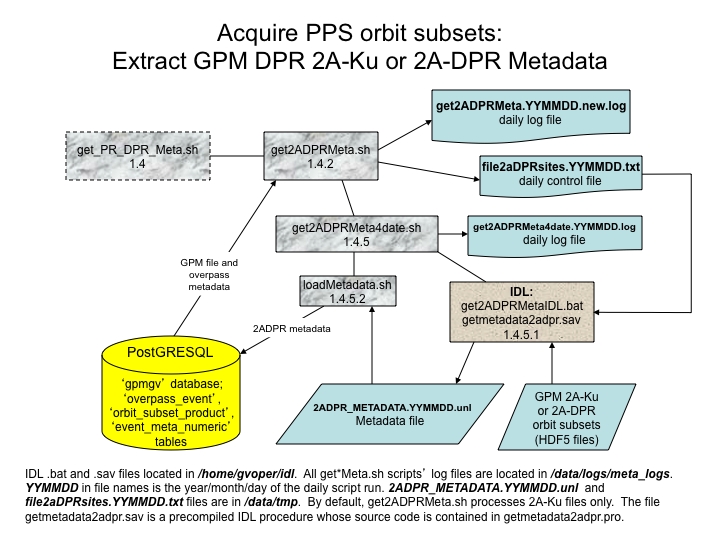


Figure 7. Schematic of the get2ADPRMeta.sh. metadata extraction script for GPM DPR data.

## wgetCTdailies.sh (RidgeMosaicCTMatch.sh)

Once-daily, satellite-specific CT files are acquired from the password-protected PPS ftp site by the script wgetCTdailies.sh (Fig. 7), which runs once per day via crontab. CT files have the naming convention CT.SAT.yyyymmdd.jjj.txt, where SAT is the satellite ID; yyyymmdd is the year, month, and day; and jjj is the day-of-year of the site overpass events tabulated in the file. Unlike orbit\_subset and ftp\_url file types, the exact name and number of daily CT files produced by PPS is known, so wgetCTdailies.sh tries to acquire specific files from the PPS.

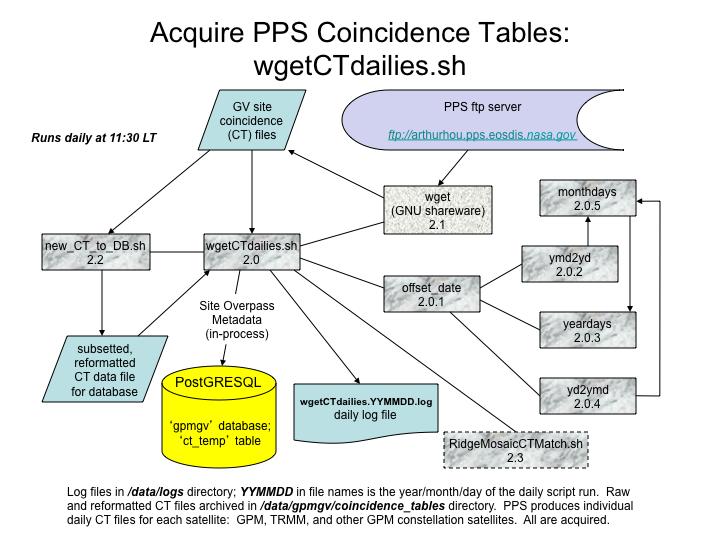


Figure 8. Schematic of processes and input/output data sources under the wgetCTdailies.sh script.

### Logic outline of wgetCTdailies.sh

Most of the logic in wgetCTdailies.sh is related to determining the list of YYYYMMDD values of CT files that need to be acquired. PPS normally produces the CT file one or more days after the fact depending on the satellite, so wgetCTdailies.sh attempts to acquire the CT file for day N-1, N-2, N-3, etc., on day N. The logic of wgetCTdailies.sh is as follows:

1. Compute the current date and time (UTC) from the system clock
2. If current time > 1503 UTC, then offset the current date by -1 days, using the script offset\_date. Otherwise, offset the current date by -2 days. The resulting date ‘ctdate’ will be the date of the current CT file to be acquired.
3. Trim the date from YYYYMMDD as used in the CT filename to the YYMMDD format.
4. Set up a loop over the list of satellites for which the PPS computes CT files, and repeat steps 5-14 for each satellite
5. Check the appstatus table in the database for the status of the current CT download for this satellite using the datestamp attribute value ‘YYMMDD’, and the app\_id attribute value ‘wgetCTSAT’, where SAT is the ID of the satellite. If no row exists in the appstatus table, then initialize one for YYMMDD with a status of ‘UNTRIED’. (*NOTE: Unlike other scripts, the datestamp attribute in the appstatus table relates to and tracks acquisition status of the date of the data file, not the date when the script is run.*)
6. Check the status of prior YYMMDD dates in the appstatus table for wgetCTSAT. If any indicate MISSING or UNTRIED, then write these YYMMDD dates to a file holding the list of YYMMDD dates whose CT files are to be acquired for this satellite, and increment the ‘ntries’ (number of download attempts) attribute values for these YYMMDDs by one in the appstatus table.
7. Check for date gaps between the date of the current CT file to be processed (‘ctdate’ from step 2) and the most recent datestamp in the appstatus table for app\_id ‘wgetCTSAT’. If there are any date gaps, compute the YYMMDD values for these dates, initialize rows for them in the appstatus table in the database, and add them to the file holding the list of CT file dates to be acquired.
8. If the status for the current datestamp ‘YYMMDD’ and app\_id ‘wgetCTSAT’ in the appstatus table indicates ‘SUCCESS’, and there are no prior CT dates to be acquired, then just report the up-to-date state and jump to the next satellite to process. Otherwise…
9. If there are prior YYMMDDs to download, then step through the list of dates in the file of prior dates to download. Build the date-specific destination file paths ‘${CT\_DATA}/${daydir}’ where $daydir is the YYYY/MM/DD path derived from the YYMMDD values, using ‘mkdir –p ${CT\_DATA}/${daydir}’ to create any of these subdirectories if they do not already exist. Then build the satellite- and date-specific CT file pathnames ‘${CT\_DATA}/${daydir}/CT.SAT.yyyymmdd.jjj.txt’ and ‘${CT\_DATA}/${daydir}/CT.SAT.yyyymmdd.jjj.unl’ for each YYMMDD and check to make certain that the ‘.txt’ file is not already present in the VN file system.
10. If the prior-date file CT file is present, mark the status of YYMMDD as ‘DUPLICATE’ in the appstatus table, and check for the presence of the postprocessed CT file ‘CT.SAT.yyyymmdd.jjj.unl’ (data subsetted and formatted for loading to the database) in the VN file system. If the ‘.unl’ file is not present, report an error situation and exit the script. Otherwise…
11. If the prior file ‘CT.SAT.yyyymmdd.jjj.txt’ is not present, format and execute a ***wget*** command to acquire the file from the PPS ftp site.
12. Check the VN file system to see whether the prior CT file was acquired. If not, then reset a flag variable ‘deletemove’ value from ‘d’ (delete) to ‘m’ (move) for later use by RidgeMosaicCTMatch.sh. If the CT file was successfully acquired, then add its pathname and the corresponding ‘.unl’ file pathname to a temporary file $FILES2DO, and set the datestamp’s status to ‘SUCCESS’ in the appstatus table.
13. Once the list of prior dates to be acquired is exhausted, check the appstatus table for rows with a status of ‘MISSING’. If the ntries attribute for any of these datestamps is 5 or greater, then set their status attribute value to ‘FAILED’ (indicating we won’t make any further attempt to acquire the CT for this YYMMDD).
14. Attempt to acquire the CT for the current YYMMDD if it does not already indicate a status of SUCCESS (step 4). Repeat the logic in steps 8-11, but for the date of the current CT file, and repeating the wget attempt (step 10) up to 2 times in a loop, waiting 10 seconds between attempts.
15. Check the file $FILES2DO holding the list of CT files to be postprocessed. If there are no entries in the file, then report this and exit the script. Otherwise…
16. For each file in $FILES2DO, call the script new\_CT\_to\_DB.sh to postprocess the CT file into a format that allows the data to be loaded into the gpmgv PostGRESQL database, and write the reformatted data to the file ‘CT.SAT.yyyymmdd.jjj.unl’. Execute a ‘***psql***’ command lo load the data from the ‘CT.SAT.yyyymmdd.jjj.unl’ file to a holding table ‘ct\_temp’ in the gpmgv database.
17. Call the script RidgeMosaicCTMatch.sh to match up/clean up Ridge mosaic files, and load coincident event and mosaic metadata to overpass\_event and coincident\_mosaic database tables.

**Table 5. Locally-configured variables defined in wgetCTdaily.sh.**

| **Variable name** | **Baseline value** | **Description** |
| --- | --- | --- |
| GV\_BASE\_DIR | /home/gvoper | As in Table 3 |
| DATA\_DIR | /data/gpmgv | As in Table 3 |
| TMP\_DIR | /data/tmp | As in Table 3 |
| CT\_DATA | ${DATA\_DIR}/coincidence\_tables | Directory where Coincidence Table (CT) files are stored in the VN filesystem. |
| BIN\_DIR | ${GV\_BASE\_DIR}/scripts | As in Table 3 |
| LOG\_DIR | /data/logs | As in Table 3 |
| LOG\_FILE | ${LOG\_DIR}/wgetCTdailies.${rundate}.log | As in Table 3, but for wgetCTdailies.sh |
| ZZZ | 10 | Number of seconds to ‘sleep’ between unsuccessful download attempts of current CT file. |
| DBTEMPFILE | ${TMP\_DIR}/dbtempfile | As in Table 3, but for wgetCTdailies.sh |
| FILES2DO | ${TMP\_DIR}/CTsToGet | Pathname to file holding a list of new CT files downloaded in this run |

The child script new\_CT\_to\_DB.sh identifies CT entries for stations in the VN (listed in Table 1), and reformats the resulting subset of CT data into a delimited, ASCII text file in a format suitable for loading into a PostGRESQL data table. The reformatted and subsetted data are written to the file CT.SAT.yyyymmdd.jjj.unl and are loaded into the ct\_temp table in the database to await additional processing.

### Logic outline of RidgeMosaicCTMatch.sh

RidgeMosaicCTMatch.sh (Fig. 9) performs two basic functions. It handles the final processing of the CT event data in the database, and cleans up the set of recently-acquired RIDGE mosaics to retain only those mosaics coincident in time with the GPM and TRMM overpasses of the VN sites, as tallied in the CT files. Most of the functionality of RidgeMosaicCTMatch.sh is encapsulated in the SQL commands in the new\_mosaicCTmatch.sql file. The logic of new\_mosaicCTmatch.sql may be summarized as follows:

1. select entries in the ct\_temp table for overpass events where the closest approach is within 250 km of the sites; identifies overpass events that are ‘new’ to the VN database;
2. insert entries for any new VN site overpass events into the overpass\_event permanent table, and removes the temporarily-held overpass event data from the ct\_temp table.
3. determine the time period between the earliest and latest site overpass events for each orbit, and expand this time period by 7”30” before to 3’30” after the original bounds
4. query the heldmosaic table to find the names of the ‘coincident mosaics’ (those with valid times within the expanded overpass event period), and write these filenames to the file mosaic2sav.lis
5. query the heldmosaic table to find the names of those mosaics with valid times outside, and prior to, the expanded overpass event period, and write these filenames to the file mosaic2del.lis
6. insert entries for the coincident mosaics into the coincident\_mosaic table, and delete entries for non-coincident mosaic times from the heldmosaic table
7. delete all the rows from the ct\_temp table now that the CT data have been processed

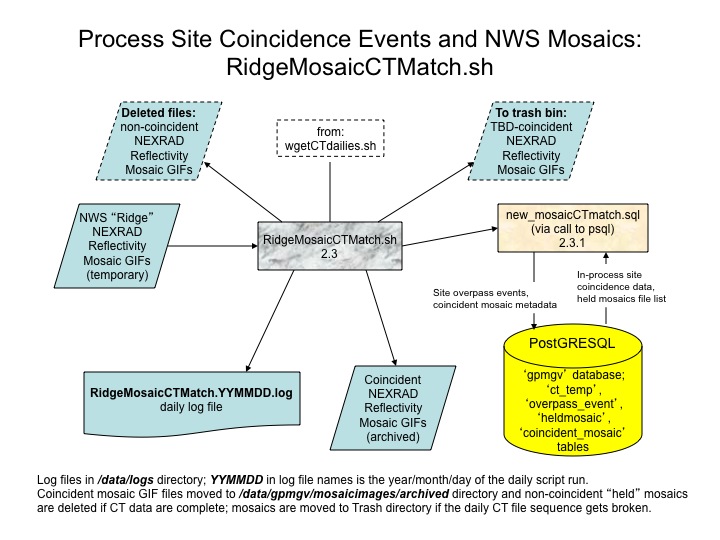


Figure 9. Schematic of RidgeMosaicCTMatch.sh script processes and input/output data.

RidgeMosaicCTMatch.sh then moves ‘coincident mosaic’ images listed in the mosaic2sav.lis file from the holding/ directory to the archivedmosaic/ directory, and depending on the value of the ‘deletemove’ parameter computed in wgetCTdailies.sh, either deletes the non-coincident mosaic images listed in the mosaic2del.lis file from the holding/ directory (normal case), or moves them to the ‘trashedmosaic’ directory where they can be evaluated later once any problems with CT file acquisition are resolved.

**Table 6. Locally-configured variables defined in RidgeMosaicCTMatch.sh.**

| **Variable name** | **Baseline value** | **Description** |
| --- | --- | --- |
| GV\_BASE\_DIR | /home/gvoper | As in Table 3 |
| DATA\_DIR | /data/gpmgv | As in Table 3 |
| TMP\_DIR | /data/tmp | As in Table 3 |
| MOSAIC\_BASE\_DATA | ${DATA\_DIR}/mosaicimages | Data directory under which archived RIDGE mosaic images are located. |
| MOSAIC\_ARCHIVE | ${MOSAIC\_BASE\_DATA}/archivedmosaic | Directory where RIDGE mosaic images whose times match up to orbit overpasses are permanently stored. |
| MOSAIC\_TMP\_DATA | ${DATA\_DIR}/mosaicimages | Common data directory under which in-process RIDGE mosaic images are held. Process-state subdirectories (e.g., /holding/) exist beneath this directory. |
| MOSAIC\_TRASH | ${MOSAIC\_TMP\_DATA}/trashedmosaic | Directory where RIDGE mosaic images whose times relative to orbit overpasses were not able to be determined are stored for later evaluation. |
| MOSAIC\_DATA\_HOLD | ${MOSAIC\_TMP\_DATA}/holding | Directory into which renamed and timestamped RIDGE mosaic files are copied and held, and also where temporary files mosaic2sav.lis and mosaic2del.lis are created by new\_mosaicCTmatch.sql commands. |
| BIN\_DIR | ${GV\_BASE\_DIR}/scripts | As in Table 3 |
| SQL\_BIN | ${BIN\_DIR}/new\_mosaicCTmatch.sql | Full pathname to file holding SQL queries/commands to be executed by the psql utility for PostGRESQL. |
| LOG\_DIR | /data/logs | As in Table 3 |
| LOG\_FILE | ${LOG\_DIR}/RidgeMosaicCTMatch.${rundate}.log | As in Table 3, but for RidgeMosaicCTMatch.sh |

### Database usage

The wgetCTdailies.sh script uses the ‘gpmgv’ database tables ‘appstatus’ and ‘ct\_temp’. The appstatus table is used to track the status of the daily runs of wgetCTdailies.sh and other scripts. Rows in the table pertaining to wgetCTdailies.sh are identified by an ‘app\_id’ attribute value of ‘wgetCT${sat}’ where ${sat} is the satellite ID associated with the specific CT file type. Details of the various status values used in the script have been described in the logic outline in Section 4.4.1. Reformatted and subsetted Coincidence Table data are written to a file by new\_CT\_to\_DB.sh, and are then loaded into the ‘ct\_temp’ table by wgetCTdailies.sh.

The RidgeMosaicCTMatch.sh script uses the ‘gpmgv’ database tables ‘coincident\_mosaic’, ‘ct\_temp’, ‘heldmosaic’, and ‘overpass\_event’, as described in the logic outline of the script presented in Section 4.4.2.

## wgetRidgeMosaic-latest.sh

The wgetRidgeMosaic-latest.sh script (Fig. 10) acquires national-scale radar reflectivity mosaic GIF images from the NWS web site at http://radar.weather.gov/ridge/Conus/RadarImg. RIDGE mosaics are produced every 10 minutes in near-real-time, and only the most recent image file (named latest.gif) is retained on the web site. The script runs every 6 minutes under the crontab to make sure that all new mosaics are acquired.

The logic of wgetRidgeMosaic-latest.sh is simple. The script calls wget with the necessary parameters and acquires the latest.gif file from the NWS web site if it is newer than the one last acquired. Any newly acquired mosaic file is copied to a timestamped file name in a “holding” directory, and is cataloged in the VN database in the heldmosaic table to await postprocessing.

**Table 7. Locally-configured variables defined in wgetRidgeMosaic-latest.sh.**

| **Variable name** | **Baseline value** | **Description** |
| --- | --- | --- |
| GV\_BASE\_DIR | /home/gvoper | As in Table 3 |
| DATA\_DIR | /data/gpmgv | As in Table 3 |
| TMP\_DIR | /data/tmp | As in Table 4 |
| MOSAIC\_TMP\_DATA | ${TMP\_DIR}/mosaicimages | Common data directory under which RIDGE mosaic images are located. Process-state subdirectories (e.g., /holding/) exist beneath this directory. |
| MOSAIC\_DATA\_HOLD | ${MOSAIC\_TMP\_DATA}/holding | Directory into which renamed and timestamped RIDGE mosaic files are copied and held. |
| MOSAIC\_DATA | ${MOSAIC\_TMP\_DATA}/radar.weather.gov/ridge/Conus/RadarImg | Directory into which RIDGE mosaic files are originally downloaded, and which mirrors the web site directory structure. The script creates this directory if it does not already exist. |
| BIN\_DIR | ${GV\_BASE\_DIR}/scripts | As in Table 3 |
| HELDTEMPFILE | ${TMP\_DIR}/wgetRidgeMosaic\_dbtempfile | Backup file to which data intended for the database are written only in cases where the database is not available. |
| LOG\_DIR | /data/logs | As in Table 3 |
| LOG\_FILE | ${LOG\_DIR}/wgetRidgeMosaic-latest.${rundate}.log | As in Table 3, but for wgetRidgeMosaic-latest.sh |

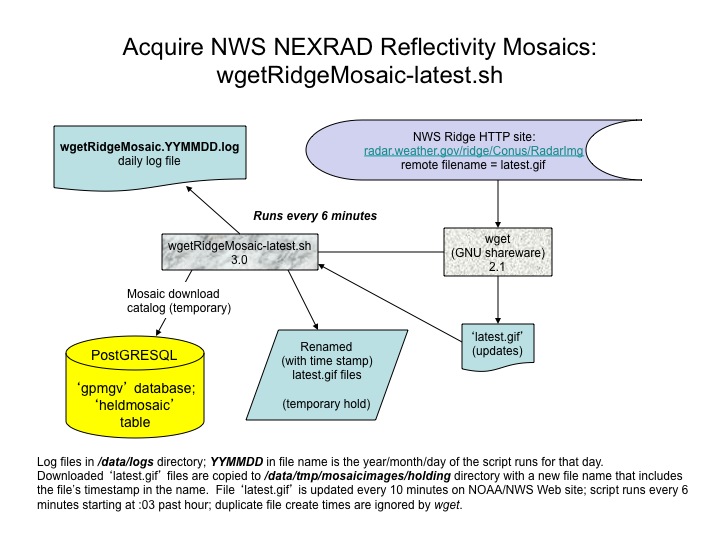


Figure 10. Schematic of wgetRidgeMosaic-latest.sh processes and input/output data sources.

## catalogQCradar.sh

As described in Section 2.2, PMM GV pushes the ground radar data files which have undergone manual quality control into the /data/gpmgv/gv\_radar/finalQC\_in directory on the GPM VN data server. Beneath this top-level directory are subdirectories for site ID, product type, and date. These directory structures are documented in the Section 4 of the GPM Validation Network Data User’s Guide. PMM GV pushes data to the GPM VN at their convenience.

Within the VN data acquisition software suite, the script catalogQCradar.sh (Fig. 11) runs once a day under the crontab to identify and catalog all new raw and quality-controlled files pushed by PMM GV into the /data/gpmgv/gv\_radar/finalQC\_in directory for the ground radar sites. The script uses a brute-force method to determine whether there are any new files to be cataloged. It first determines which years’ data are to be considered in looking for new files. If within 30 days of the new year, both the current year and the prior year will be considered, otherwise only the current year will be considered in the recursive listings to be retrieved. It then:

1. Deletes the previous run’s database-ready file ‘finalQC\_KxxxMeta.unl’ if it exists.
2. Gets the site IDs (subdirectories) under the /data/gpmgv/gv\_radar/finalQC\_in directory. For each site ID, the script:
   1. Loops over the list of years determined at the start of the script. For each year “YYYY”, the script:
      1. renames the previous run’s directory listings files ‘lsKXXXfinal\_YYYY.new’ to ‘lsKXXXfinal\_YYYY.old’, where KXXX is the site ID of the ground radar and YYYY is the 4-digit year for the data listed. If no previous ‘lsKXXXfinal\_YYYY.new’ file exists, an empty ‘lsKXXXfinal\_YYYY.old’ file is created
      2. retrieves a unix recursive listing (ls –R) of all files under that site subdirectory for year YYYY, processes the output from the recursive file listing command to merge the path information with the names of the files under that directory path, and writes the file pathnames to the plain text file ‘lsKXXXfinal\_YYYY.new’
      3. does a unix diff (difference) between the ‘.old’ and ‘.new’ file pairs, and pipes the results to ‘lsKXXXfinal\_YYYY.diff’
      4. steps through the ‘.diff’ file line by line and identifies additions to the ‘.new’ file listing relative to the previous file listing in ‘.old’
      5. passes the new file pathname to the internal function ‘dbfileprep’ to prepare metadata fields for loading into the 'gpmgv' database, and appends the metadata to the delimited ASCII text file ‘finalQC\_KxxxMeta.unl’ (a literal file name, holding data for all site IDs/years for this run).
3. Checks the status of the file ‘finalQC\_KxxxMeta.unl’ and, if it is of non-zero size, the SQL command in the file catalogQCradar.sql is executed within the ‘psql’ database utility to load the file’s data into the ‘gpmgv’ database.

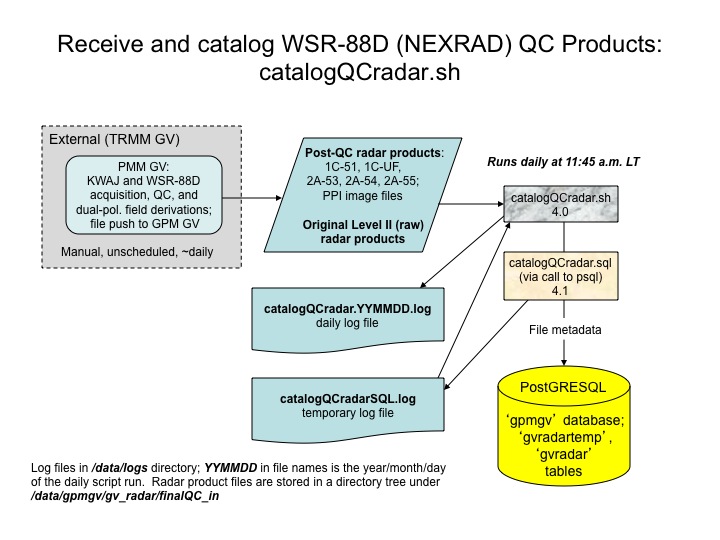


Figure 11. Schematic of catalogQCradar.sh script processes and data stores/sources.

### Database usage

Once the new file metadata for all site IDs have been written to finalQC\_KxxxMeta.unl by the catalogQCradar.sh script, the SQL commands in the file catalogQCradar.sql are executed by calling the psql utility from the script. The SQL commands load the data in the delimited file into the table 'gvradartemp' in the 'gpmgv' PostGRESQL database, identifies any duplicate entries between the filenames listed in the 'gvradartemp' table and the existing file catalog in the permanent ‘gvradar’ table (i.e., files erroneously identified as new), inserts the non-duplicate data for new files into the ‘gvradar’ table, and deletes the rows from the 'gvradartemp' table.

**Table 8. Locally-configured variables defined in catalogQCradar.sh.**

| **Variable name** | **Baseline value** | **Description** |
| --- | --- | --- |
| GV\_BASE\_DIR | /home/gvoper | As in Table 3 |
| DATA\_DIR | /data/gpmgv | As in Table 3 |
| TMP\_DIR | /data/tmp | As in Table 4 |
| QCDATADIR | ${DATA\_DIR}/gv\_radar/finalQC\_in | Common data directory under which quality-controlled radar data files are stored. Site/type/date subdirectory structures exist beneath this directory. |
| BIN\_DIR | ${GV\_BASE\_DIR}/scripts | As in Table 3 |
| SQL\_BIN | ${BIN\_DIR}/catalogQCradar.sql | Full pathname to file holding SQL queries/commands to be executed by the psql utility for PostGRESQL. |
| loadfile | ${TMP\_DIR}/finalQC\_KxxxMeta.unl | File to hold formatted metadata ready for loading into the ‘gpmgv’ database table ‘gvradartemp’. ***This file pathname must be defined identically in both the script catalogQCradar.sh and the SQL command file catalogQCradar.sql*** |
| LOG\_DIR | /data/logs | As in Table 3 |
| LOG\_FILE | ${LOG\_DIR}/catalogQCradar.${rundate}.log | As in Table 3, but for catalogQCradar.sh |
| DB\_LOG\_FILE | ${LOG\_DIR}/catalogQCradarSQL.log | Log file to hold output from psql for the SQL queries in catalogQCradar.sql |

## wget\_GT7\_GPM.sh (extract\_daily\_predicts.sh)

The script wget\_GT7\_GPM.sh (Fig. 12) downloads daily files containing 7-day predictions of the GPM orbit track at 1-second intervals from the PPS ftp site, and runs once per day via crontab. CT files have the naming convention **GT-7.SAT.yyyymmdd.jjj.txt**, where **SAT** is the satellite ID (only GPM used); **yyyymmdd** is the year, month, and day; and **jjj** is the day-of-year of the start of the predictions tabulated in the file. Like the CT file types, the exact name and number of daily GT-7 files produced by PPS is known, so wget\_GT7\_GPM.sh tries to acquire specific files from the PPS. If the script succeeds in downloading one or more daily 7-day prediction files, it runs the child script extract\_daily\_predicts.sh to extract one day’s worth of prediction data from the 7-day files and saves these data to daily 1-day prediction files with no overlaps in time coverage. The extracted daily data are written to files with the naming convention **GPM\_1s\_subpts.yyyymmdd.txt**, where **yyyymmdd** is the year, month, and day of the prediction data tabulated in the file. In the normal case where a GT-7 file is available for every day, only the first 24 hours of data are extracted from the GT-7 file to produce the corresponding GPM\_1s\_subpts file for a date. In the presence of a gap of more than one day between available GT-7 files, then prediction data for a GPM\_1s\_subpts file for a date will be extracted from later times in the nearest prior GT-7 data file with prediction data for the date.

### Logic outline of wget\_GT7\_GPM.sh

Most of the logic in wget\_GT7\_GPM.sh is related to determining the list of YYYYMMDD values of GT-7 files that need to be acquired. PPS normally produces the GT-7 file on or before the datestamped date, but wgetCTdailies.sh attempts to acquire the CT file for day N-1 on day N. The logic of wget\_GT7\_GPM.sh is as follows:

1. Set up configuration environment variables, daily script log file, and temporary file pathnames. Delete the existing temporary file $FILES2DO if it exists from a previous run.
2. Compute the current date and time (UTC) from the system clock.
3. If current time > 1503 UTC, then offset the current date by -1 days, using the script offset\_date. Otherwise, offset the current date by -2 days. The resulting date ‘ctdate’ will be the date of the current CT file to be acquired.
4. Trim the date from YYYYMMDD as used in the GT-7 filename to the YYMMDD format for use in database tracking.
5. Set up a loop over the list of satellites for which the PPS computes GT-7 files, and repeat steps 6-17 for each satellite (currently hard-wired to GPM only).
6. Check the appstatus table in the database for the status of the current GT-7 download for this satellite using the datestamp attribute value ‘YYMMDD’, and the app\_id attribute value ‘wgetGTSAT’, where SAT is the ID of the satellite. If no row exists in the appstatus table, then initialize one for YYMMDD with a status of ‘UNTRIED’. (*NOTE: Unlike other scripts, the datestamp attribute in the appstatus table relates to and tracks acquisition status of the date of the data file, not the date when the script is run.*)
7. Check the status of prior YYMMDD dates in the appstatus table for wgetGTSAT. If any indicate MISSING or UNTRIED, then write these YYMMDD dates to a file holding the list of YYMMDD dates whose GT-7 files are to be acquired for this satellite, and increment the ‘ntries’ (number of download attempts) attribute values for these YYMMDDs by one in the appstatus table.
8. Check for date gaps between the date of the current GT-7 file to be processed (‘ctdate’ from step 2) and the most recent datestamp in the appstatus table for app\_id ‘wgetGTSAT’. If there are any date gaps, compute the YYMMDD values for these dates, initialize rows for them in the appstatus table in the database, and add them to the file holding the list of GT-7 file dates to be acquired.
9. If the status for the current datestamp ‘YYMMDD’ and app\_id ‘wgetGTSAT’ in the appstatus table indicates ‘SUCCESS’, and there are no prior GT-7 dates to be acquired, then just report the up-to-date state and jump to the next satellite to process. Otherwise…
10. If there are prior YYMMDDs to download, then step through the list of dates in the file of prior dates to download. For each YYMMDD, build the year-specific destination directory name and create it as needed, and build the satellite- and date-specific GT-7 file pathname ‘${GT\_DATA}/${yeardir}/GT-7.SAT.yyyymmdd.jjj.txt’ and check to make certain that the file is not already present in the VN file system.
11. If the prior-date file GT-7 file is present, mark the status of YYMMDD as ‘DUPLICATE’ in the appstatus table. Otherwise…
12. If the prior file ‘GT-7.SAT.yyyymmdd.jjj.txt’ is not present, format and execute a ***wget*** command to acquire the file from the PPS ftp site.
13. Check the VN file system to see whether the prior GT-7 file was acquired. If yes, then write the file’s pathname to a temporary file $FILES2DO and set the datestamp’s status to ‘SUCCESS’ in the appstatus table.
14. Once the list of prior dates to be acquired is exhausted, check the appstatus table for rows with a status of ‘MISSING’. If the ntries attribute for any of these datestamps is greater than 9, then set their status attribute value to ‘FAILED’ (indicating we won’t make any further attempt to acquire the GT-7 for this YYMMDD).
15. Attempt to acquire the GT-7 for the current YYMMDD if it does not already indicate a status of SUCCESS (step 4). Repeat the logic in steps 10-13, but for the date of the current GT-7 file, and repeating the wget attempt (step 10) up to 2 times in a loop, waiting 1 seconds between attempts.
16. Check the file $FILES2DO holding the list of GT-7 files to be post-processed. If there are no entries in the file, then report this and exit the script. Otherwise…
17. Call the script extract\_daily\_predicts.sh with the argument $FILES2DO, to post-process the GT-7 file(s) into non-overlapping 1-day prediction files.

### Logic outline of extract\_daily\_predicts.sh

Most of the logic in extract\_daily\_predicts.sh is related to determining the list of YYYYMMDD values of GPM\_1s\_subpts.YYYYMMDD.txt files that need to be created from data contained in the GT-7 files. The logic of extract\_daily\_predicts.sh is as follows:

1. Set up configuration environment variables.
2. Compute current year (UTC) from system clock.
3. Find the datestamp of the most recent GPM\_1s\_subpts.YYYYMMDD.txt file
4. If no GT-7 file listing is specified as the argument to the script, then as a substitute, get a listing of all GT-7 files for the current year contained in the VN file system.
5. Find the name and datestamp of the oldest GT-7 file in the listing from (4).
6. Loop through all the GT-7 files in the listing from (4), from the oldest to the newest. For each file, repeat steps 7-10.
7. Determine the date gap between the latest 1-day file and the GT-7 file.
8. If the date gap is negative or zero (GPM\_1s\_subpts datestamp the same or later than the GT-7 file datestamp), then print a message that this GT-7 file is old and being ignored, and skip to the next GT-7 file in the loop.
9. If the date gap is exactly 1 day, then set the datestamp of the new GPM\_1s\_subpts file to the datestamp of the GT-7 file, compose the pathname of a new GPM\_1s\_subpts file, find the rows in the GT-7 file whose date parts match the datestamp date, write them to the new GPM\_1s\_subpts file, and skip to the next GT-7 file in the loop.
10. If the date gap is greater than one day, then for each date in the resulting gap:
    1. Increment the date of the latest 1-day file by one day and compute a new date gap between it and the 7-day file’s date.
    2. If the new date gap is less than 7 days, then set the datestamp of the new GPM\_1s\_subpts file to the incremented datestamp, compose the pathname of a new GPM\_1s\_subpts file, find the rows in the GT-7 file whose date parts match the incremented datestamp date, and write them to the new GPM\_1s\_subpts file.
    3. If the date gap is 7 days or greater, then write an error message indicating that no data are available for the date of the incremented datestamp. ***This situation indicates a failure in downloading new GT-7 files from the PPS and needs to be immediately addressed by the PPS and/or VN administrators.***
    4. Set the new date gap for the test in (10) to the difference between the current GT-7 file datestamp and the incremented 1-second datestamp (from 10.a)

### Database usage

The wget\_GT7\_GPM.sh script uses the ‘gpmgv’ database table ‘appstatus’. The appstatus table is used to track the status of the daily runs of wget\_GT7\_GPM.sh and other scripts. Rows in the table pertaining to wget\_GT7\_GPM.sh are identified by an ‘app\_id’ attribute value of ‘wgetGT${sat}’ where ${sat} is the satellite ID associated with the specific GT-7 file type. Details of the various status values used in the script have been described in the logic outline in Section 4.7.1. The extract\_daily\_predicts.sh script does not interact with the gpmgv database.

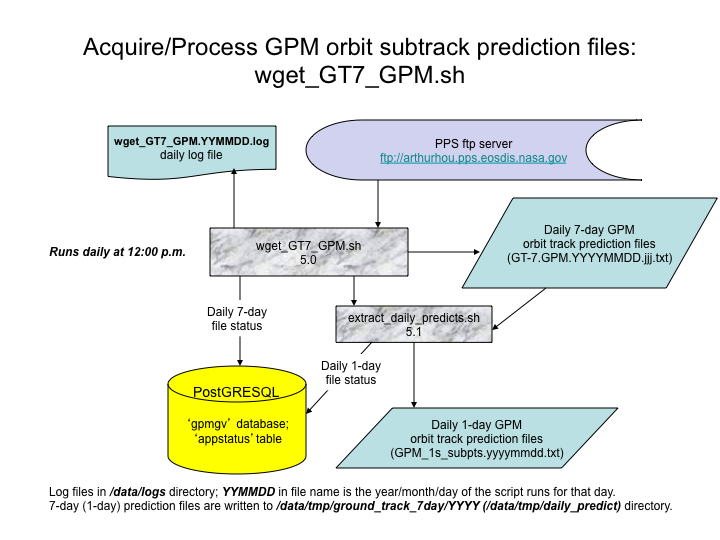


Figure 12. Schematic of wget\_GT7\_GPM.sh script processes and data stores/sources.

**Table 9. Locally-configured variables defined in wget\_GT7\_GPM.sh.**

| **Variable name** | **Baseline value** | **Description** |
| --- | --- | --- |
| GV\_BASE\_DIR | /home/gvoper | As in Table 3 |
| DATA\_DIR | /data/tmp | As in Table 3 |
| TMP\_DIR | /data/tmp | As in Table 4 |
| GT\_DATA | ${DATA\_DIR}/ground\_track\_7day | Top-level directory where downloaded GT-7 files are stored under year-specific subdirectories (is external to the VN filesystem). |
| BIN\_DIR | ${GV\_BASE\_DIR}/scripts | As in Table 3 |
| LOG\_DIR | /data/logs | As in Table 3 |
| LOG\_FILE | ${LOG\_DIR}/wget\_GT7\_GPM.${rundate}.log | As in Table 3, but for wget\_GT7\_GPM.sh |
| ZZZ | 1 | Number of seconds to ‘sleep’ between unsuccessful download attempts of current GT-7 file. |
| DBTEMPFILE | ${TMP\_DIR}/dbtempfile | As in Table 3, but for wget\_GT7\_GPM.sh |
| FILES2DO | ${TMP\_DIR}/GTsToGet | Pathname to file holding a list of new GT-7 files downloaded in this run |
| USERPASS | (private) | PPS username and password (same) for authorized user ftp login |
| FIXEDPATH | ftp://arthurhou.pps.eosdis.nasa.gov/gpmdata/coincidence | ftp URL comprising the “common” location under which the GT-7 daily data files are located in a YYYY/MM/DD date-specific subdirectory structure. |

**Table 10. Locally-configured variables defined in extract\_daily\_predicts.sh.**

| **Variable name** | **Baseline value** | **Description** |
| --- | --- | --- |
| GV\_BASE\_DIR | /home/gvoper | As in Table 3 |
| BIN\_DIR | ${GV\_BASE\_DIR}/scripts | As in Table 3 |
| PREDICT\_DIR | /data/tmp/ground\_track\_7day/$yearnow | Year-specific directories where GT-7 files are stored (is external to the VN filesystem). The $yearnow part of the path is internally computed in the script from the system clock. Must be configured the same as GT\_DATA in Table 9. |
| TARGET\_DIR | /data/tmp/daily\_predict | Directory to which the computed output GPM\_1s\_subpts.YYYYMMDD.txt files are written. |

## wget\_orbdef\_GPM.sh (extract\_orbits)

The script wget\_orbdef\_GPM.sh (Fig. 13) downloads single-orbit files containing GPM orbit number and start and end date/times from the PPS ftp site, and runs once per day via ***crontab***. Orbit definition files have the naming convention:

**GPMCORE.yyyymmdd.hhmmssddd\_YYYYMMDD.HHMMSSDDD.001.ORBDEF.txt**

where:

**GPMCORE** - literal characters 'GPMCORE'

**SSSS** - ID of the satellite ('GPM' only for this script)

**yyyymmdd** - year (4-digit), month, and day of the orbit start

**hhmmssddd** - hour, minute, second, and decimal second of orbit start

**YYYYMMDD** - year (4-digit), month, and day of the orbit end

**HHMMSSDDD** - hour, minute, second, and decimal second of orbit end

**ORBDEF.txt** - literal characters 'ORBDEF.txt'

The exact name and number of ORBDEF files produced each day by PPS varies by the GPM orbit configuration, so wget\_orbdef\_GPM.sh uses a set of ***wget*** options configured to mirror the set of recent ORBDEF.txt files present on the PPS ftp site. If the script succeeds in downloading one or more ORBDEF files, it calls the internal function extract\_orbits for each file to extract the orbit number and start and end times from the ORBDEF file, format the times into a PostGRESQL “DATESTAMP WITH TIME ZONE” format, and load these values into the ‘gpm\_orbits’ table in the ‘gpmgv’ database.

### Logic outline of wget\_orbdef\_GPM.sh

Most of the logic in wget\_orbdef\_GPM.sh is related to determining the list of YYYY/MM values of the directories in the PPS repository files that need to be mirrored. The logic of wget\_orbdef\_GPM.sh is as follows:

1. Set up configuration environment variables, daily script log file, and temporary file pathnames. Delete the existing temporary file $FILES2DO if it exists from a previous run.
2. Compute the current date and time (UTC) from the system clock.
3. Offset the current date by -2 days, using the script offset\_date. The resulting date ‘ctdate2’ is trimmed to 6 digits (yyyymm) to be used to determine the latest year and month directory components of the current ORBDEF files to be mirrored.
4. Offset the current date by -28 days, using the script offset\_date. The resulting date ‘ctdate1’ is trimmed to 6 digits (yyyymm) to be used to determine the earliest year and month directory components of the ORBDEF files to be mirrored
5. Set up a loop over the list sorted, unique yyyymm values defined by ctdate1 and ctdate2.
6. Generate the subdirectory value yyyy/mm from the unique yyyymmdd values.
7. If it doesn’t yet exist, create the local directory yyyy/mm under the common data directory $GEO\_DATA defined in step 1.
8. Format and execute a ***wget*** command to mirror the ORBDEF.txt files and their date-specific subdirectories from the PPS ftp site to the local subdirectory from (7).
9. Check the resulting file listing from the wget utility to see whether any new ORBDEF.txt files were acquired. If yes, then write the file pathname(s) to a temporary file $FILES2DO.
10. Check the file $FILES2DO holding the list of downloaded ORBDEF.txt files to be post-processed. If there are no entries in the file, then report this and exit the script. Otherwise…
11. For each file listed in $FILES2DO, call the internal function extract\_orbits.sh with the argument $FILES2DO, to post-process the ORBDEF.txt file(s) and load their orbit definition information to the ‘gpmgv’ database.

### Logic outline of extract\_orbits()

The logic of the function extract\_orbits() is as follows:

1. Set up a temporary file in the location defined by the environment variable TMP\_DIR.
2. Run the contents of the ORBDEF.txt through a multi-step ‘sed’ command to:
   1. Skip over the 22 lines of prologue and table headings
   2. Remove any trailing spaces in the data line
   3. Replace multiple spaces between data elements with the ‘|’ field delimiter
   4. Write the resulting delimited data line to the temporary file
3. Run the contents of the temporary data file through formatting commands to:
   1. Extract the GPM orbit number
   2. Extract the orbit start time and reformat it to a database-compatible format
   3. Extract the orbit end time and reformat it to a database-compatible format
4. Write the orbit number and its starting and ending datetimes to the ‘gpm\_orbits’ table in the ‘gpmgv’ database.

### Database usage

The extract\_orbits function contained and called in the wget\_orbdef\_GPM.sh script writes data to the ‘gpmgv’ database table ‘gpm\_orbits’.

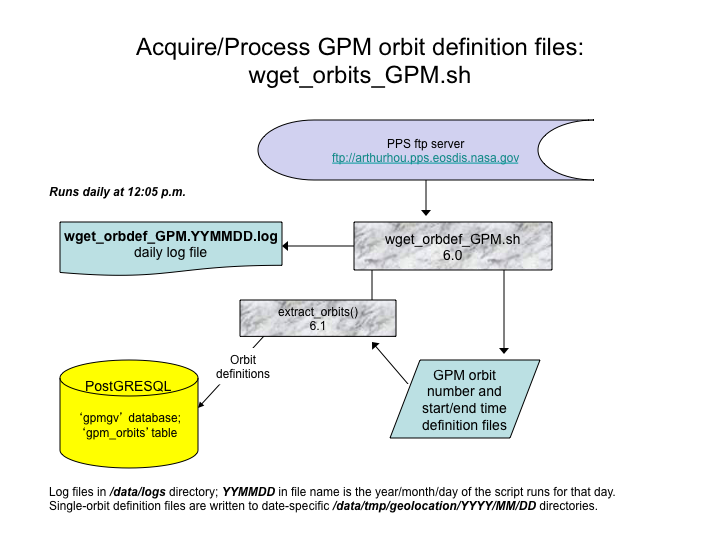


Figure 13. Schematic of wget\_orbdef\_GPM.sh script processes and data stores/sources.

**Table 11. Locally-configured variables defined in wget\_orbdef\_GPM.sh.**

| **Variable name** | **Baseline value** | **Description** |
| --- | --- | --- |
| GV\_BASE\_DIR | /home/gvoper | As in Table 3 |
| DATA\_DIR | /data/tmp | As in Table 3 |
| TMP\_DIR | /data/tmp | As in Table 4 |
| GEO\_DATA | ${DATA\_DIR}/geolocation | Top-level directory where downloaded ORBDEF.txt files are stored under year-mont-day specific subdirectories (is external to the public VN filesystem). |
| BIN\_DIR | ${GV\_BASE\_DIR}/scripts | As in Table 3 |
| LOG\_DIR | /data/logs | As in Table 3 |
| LOG\_FILE | ${LOG\_DIR}/wget\_orbdef\_GPM.${rundate}.log | As in Table 3, but for wget\_orbdef\_GPM.sh |
| TEMPFILE | ${TMP\_DIR}/GEO\_tempfile | Pathname to file holding diagnostic and file download information from ***wget*** |
| FILES2DO | ${TMP\_DIR}/GEOtoGet | Pathname to file holding a list of new ORBDEF.txt files downloaded in this run |
| USERPASS | (private) | PPS username and password (same) for authorized user ftp login |
| FIXEDPATH | ftp://arthurhou.pps.eosdis.nasa.gov/gpmdata/geolocation | ftp URL comprising the “common” location under which the ORBDEF.txt data files are located in a YYYY/MM/DD date-specific subdirectory structure. |

# The ‘gpmgv’ database in PostgreSQL

The data acquisition software of the GPM Validation Network uses a number of dynamic tables in a PostGRESQL database named ‘gpmgv’ to track the status of data acquisition and post-processing, and to catalog the data products that have been acquired. Additional static-data tables hold fixed information such as ground radar station identifiers and locations, satellite product subset identifiers, and other attributes. A listing of each table’s schema will be presented below, with a basic description of the table’s use. A basic understanding of relational database design and function is assumed. For a tutorial on databases in PostGRESQL, refer to the online tutorial found at <http://www.postgresql.org/docs/8.0/static/tutorial.html>. Only those actual tables involved in the data acquisition and preprocessing are described below. Many other tables and database views (predefined complex queries in the form of synthetic tables) are defined and populated in the ‘gpmgv’ database.

## Dynamic tables

Dynamic tables hold information that varies for each case and/or changes over time. These are the tables that are written to and updated by the VN data acquisition scripts. The dynamic tables will be presented in alphabetical order by table name.

**appstatus table –** VN data acquisition and processing scripts use the ‘appstatus’ table to track the status of their daily runs and/or the status of data acquisition. Each script that uses the table has its own ‘app\_id’ (application ID) attribute value to identify the rows that pertain to that script and/or its acquired data source. The ‘datestamp’ attribute is the date, in YYMMDD format, that pertains to the date of the script run or, for wgetCTdailies.sh or wget\_GT7\_GPM.sh, the date of the data to be acquired. Each row in the table is uniquely identified by a unique combination of ‘app\_id’ and ‘datestamp’, which together comprise the Primary Key of the table. The ‘first\_attempt’ attribute is automatically generated by the database, and holds the date and time that the row was created (i.e., when the script was first run for the given ‘app\_id’ and ‘datestamp’). The ‘ntries’ attribute represents the number of attempts that the script has made to download the data for the given ‘datestamp’, and status represents the current/last status of the data acquisition attempt. All attributes except ‘first\_attempt’ are assigned by the script pertaining to the ‘app\_id’.

Table "appstatus"

Column | Type | Modifiers

---------------+---------------+-----------------------------

app\_id | character(15) | not null

first\_attempt | date | default ('now'::text)::date

datestamp | character(6) | not null

ntries | integer | default 0

status | character(1) | default 'U'::bpchar

Indexes:

"appstatus\_pkey" primary key, btree (app\_id, datestamp)

**coincident\_mosaic table -** The coincident mosaic table holds a catalog of NWS RIDGE radar mosaic images that are coincident in time with TRMM and GPM orbits that overpass the core VN area in the continental U.S. Its columns are ‘orbit’ (orbit number) and ‘filename’ (time-stamped name assigned to RIDGE mosaic GIF images by the wgetRidgeMosaic-latest.sh script). The table is populated by the SQL commands in the new\_mosaicCTmatch.sql file, which are executed by a call to the psql utility within the script RidgeMosaicCTMatch.sh.

Table "coincident\_mosaic"

Column | Type | Modifiers

----------+-----------------------+-----------

orbit | integer | not null

filename | character varying(31) | not null

Indexes:

"coincident\_mosaic\_pkey" primary key, btree (orbit, filename)

**ct\_temp table –** The ct\_temp table holds data from a CT file, partially preprocessed to extract and format data fields listing the satellite orbit number, overpass times, and nearest approach distance to VN ground radar sites. The data are written to a delimited text file by new\_CT\_to\_DB.sh, called from wgetCTdailies.sh, which then loads the data in the file into the ‘ct\_temp’ table. The data are only stored temporarily in the ct\_temp table, pending final processing by the RidgeMosaicCTMatch.sh script, which then deletes the contents of the ‘ct\_temp’ table.

Table "ct\_temp"

Column | Type | Modifiers

---------------+--------------------------+-----------

orbit | integer |

radar\_name | character varying |

radar\_id | character varying |

overpass\_time | timestamp with time zone |

proximity | double precision |

**event\_meta\_numeric table –** The ‘event\_meta\_numeric’ table holds a variety of metadata values computed from PR, DPR and ground radar products that describe features of a site overpass event, such as the overlap between the PR/DPR swath and the ground radar coverage, and the areal amount of precipitation within the PR/GR and DPR/GR overlap area. Only one metadata variable ‘value’ is stored in each row of the table, and it is labeled by the unique ‘event\_num’ assigned to the site overpass event, and the ‘metadata\_id’ (the identifier of the variable value being stored). The meanings of the ‘metadata\_id’ integer values are defined in the ‘metadata\_parameter’ static table. Together, event\_num and metadata\_id comprise the Primary Key of the table, and each of these attributes is also a Foreign Key referencing other tables. The event\_meta\_numeric table is written to by the script loadMetadata.sh, called from the scripts get2A23Meta.sh, get2A25Meta.sh, and get2ADPRMeta4date.sh, which compute the metadata values. These metadata values are used to determine the occurrence of rain for the satellite overpass events by the volume-matching scripts, which, by default, run the SQL commands in the file rainCases100kmAddNewEvents.sql to update entries in the table ‘rainy100inside100’ (not described here).

Table "event\_meta\_numeric"

Column | Type | Modifiers

-------------+------------------+-----------

event\_num | integer | not null

metadata\_id | integer | not null

value | double precision | not null

Indexes:

"event\_meta\_numeric\_pkey" primary key, btree (event\_num, metadata\_id)

Foreign-key constraints:

"$1" FOREIGN KEY (event\_num) REFERENCES overpass\_event(event\_num)

"$2" FOREIGN KEY (metadata\_id) REFERENCES metadata\_parameter(metadata\_id)

**gpm\_orbits table** – The ‘gpm\_orbits’ table contains GPM orbit number and the orbit starting and ending datetimes extracted from GPM orbit definition files and formatted for entry to the database by the wget\_orbdef\_GPM.sh script.

Table "gpm\_orbits"

Column | Type | Modifiers

-----------+--------------------------+-----------

orbit | integer | not null

starttime | timestamp with time zone | not null

endtime | timestamp with time zone | not null

Indexes:

"gpm\_orbits\_pkey" PRIMARY KEY, btree (orbit)

**gvradar table –** The ‘gvradar’ table contains the catalog of all quality-controlled ground radar product files. *The ‘filename’ attribute itself is the Primary Key of the table, thus all QC’ed radar products are assumed, and constrained, to have unique filenames by convention.* The ‘filepath’ is the partial path to the file, immediately preceding the filename, and following the common path to all quality-controlled ground radar files in the VN file system. The ‘product’ attribute is the type of ground radar data file being cataloged, and the ‘nominal’ attribute is the starting datetime of the ground radar volume scan, typically rounded or truncated to the nearest minute. The ‘radar\_id’ attribute is the (typically 4-letter) ID of the ground radar, as previously defined as ‘instrument\_id’ in the ‘instrument’ static table, which it references as a Foreign Key. Thus, any new Radar ID to be added to the VN must first be defined as an ‘instrument\_id’ value in the ‘instrument’ table before its data can be successfully cataloged and processed by the VN software. Data rows to be added to the ‘gvradar’ table are first loaded to the ‘gvradartemp’ table, where duplicate rows are checked for and eliminated by SQL commands in the file catalogQCradar.sql prior to inserting the rows in the ‘gvradar’ table.

Table "gvradar"

Column | Type | Modifiers

----------+--------------------------+-----------

product | character varying(15) | not null

radar\_id | character varying(15) | not null

nominal | timestamp with time zone | not null

filepath | character varying(63) | not null

filename | character varying(63) | not null

fileidnum | bigint | not null default nextval('gvradar\_fileidnum\_seq')

Indexes:

"gvradar\_pkey" primary key, btree (filename)

"gvradar\_fileidnum\_key" UNIQUE, btree (fileidnum)

"gvradidx3" btree (product, radar\_id, nominal)

Foreign-key constraints:

"$1" FOREIGN KEY (radar\_id) REFERENCES instrument(instrument\_id)

**gvradartemp table –** The ‘gvradartemp’ table has the identical columns as the ‘gvradar’ table except for ‘fileidnum’. Data rows for new ground radar product files to be cataloged in the gpmgv database are first loaded to the (initially empty) ‘gvradartemp’ table, to eliminate the possibility of a data load error to the ‘gvradar’ table due to a duplicate row. Any duplicate rows between ‘gvradar’ and ‘gvradartemp’ are identified, and non-duplicate rows in ‘gvradartemp’ are loaded to ‘gvradar’. See the ‘gvradar’ table description, and the catalogQCradar.sh script description. The ‘radar\_id’ attribute is the (typically 4-letter) ID of the ground radar, and must be previously defined as ‘instrument\_id’ in the ‘instrument’ static table, which it references as a Foreign Key.

Table "gvradartemp"

Column | Type | Modifiers

----------+--------------------------+-----------

product | character varying(15) | not null

radar\_id | character varying(15) | not null

nominal | timestamp with time zone | not null

filepath | character varying(63) | not null

filename | character varying(63) | not null

Foreign-key constraints:

"$1" FOREIGN KEY (radar\_id) REFERENCES instrument(instrument\_id)

**heldmosaic table –** The heldmosaic table holds the catalog of all NWS RIDGE Mosaic images acquired in near-real-time and renamed from ‘latest.gif’ to a unique time-stamped file name by the wgetRidgeMosaic-latest.sh script. The ‘heldmosaic’ table is a temporary catalog of these products, to track them until they can be evaluated to see which products are coincident in time with TRMM site overpass events. The ‘nominal’ attribute is the datetime of the radar mosaic, rounded to the nearest minute.

Table "heldmosaic"

Column | Type | Modifiers

----------+--------------------------+-----------

nominal | timestamp with time zone |

filename | character varying |

Indexes:

"uniqfilename" unique, btree (filename)

**metadata\_temp table –** The ‘metadata\_temp’ table is the (initially empty) table to which new metadata destined for the 'event\_meta\_num' table are loaded. The script loadMetadata.sh loads data from the delimited text files into the ‘metadata\_temp’ table, checks these data rows against permanent metadata in the 'event\_meta\_num' table for duplicates, and inserts non-duplicate (new) metadata into the permanent ‘event\_meta\_num' table. It has attributes and constraints identical to the ‘event\_meta\_num' table.

Table "metadata\_temp"

Column | Type | Modifiers

-------------+------------------+-----------

event\_num | integer | not null

metadata\_id | integer | not null

value | double precision | not null

Indexes:

"metadata\_temp\_pkey" primary key, btree (event\_num, metadata\_id)

Foreign-key constraints:

"$1" FOREIGN KEY (metadata\_id) REFERENCES metadata\_parameter(metadata\_id)

"$2" FOREIGN KEY (event\_num) REFERENCES overpass\_event(event\_num)

**orbit\_subset\_product table –** The ‘orbit\_subset\_product’ table holds the catalog of all GPM, TRMM, and constellation satellite data products (either orbit-subset or full-orbit) ingested by the VN system from the PPS. The scripts get\_PPS\_CS\_data.sh (for operational data) and get\_PPS\_CS\_v4testdata.sh (for ITE test data versions) acquire the PPS products, extract most of the fields needed to catalog the products from the filenames, and insert a row for each successfully acquired product into the table. Products are uniquely identified in the table by the combination of the attributes ‘sat\_id’ (GPM, TRMM, METOPA, etc.), ‘orbit’ number, ‘product\_type’ (1C21, 2A25, etc.), ‘subset’ name (CONUS, FullOrbit, KWAJ, etc.), and product ‘version’ (6 or 7 for legacy TRMM; V04A or the like for GPM-era data), which together also comprise the Primary Key of the table. File names of the data products must be unique within a product\_type for purposes of storage and retrieval from the VN file system, but this constraint is not enforced in the orbit\_subset\_product table. The combination of the ‘sat\_id’ and ‘subset’ attributes is a Foreign Key referencing the ‘productsubset’ table, which is one of the tables required to identify the satellite product ‘subset’ (orbit subset name) which contains data for the satellite’s overpasses of a given region enclosing one or more ground radar sites (see ‘siteproductsubset’ table). Therefore, any new satellite product subsets whose data files are to be stored and cataloged must first be defined in the ‘productsubset’ table.

Table "orbit\_subset\_product"

Column | Type | Modifiers

--------------+------------------------+--------------------

sat\_id | character varying(15) | not null

orbit | integer | not null

product\_type | character varying(15) | not null

filedate | date |

filename | character varying(120) |

subset | character varying(15) | not null

version | character varying(5) | not null default 6

Indexes:

"orbit\_subset\_product\_pkey" primary key, btree (sat\_id, subset,

orbit, product\_type, "version")

Foreign-key constraints:

"$1" FOREIGN KEY (sat\_id, subset) REFERENCES productsubset(sat\_id,

subset)

**overpass\_event table –** The ‘overpass\_event’ table is the catalog of all orbital overpasses of the ground radar sites of concern to the VN. It contains data fields extracted from satellite-specific daily CT (coincidence table) files from PPS, filtered to VN sites only, and to overpasses with a nearest distance of 250 km or less. The ‘event\_num’ is the unique, serial integer identifier for the site overpass event, and is automatically assigned by the database when a row for a new event is added to the table. Data to be loaded to this table are first loaded to the ‘ct\_temp’ table, so that a failure to load new data to ‘overpass\_event’ due to an attempt to insert duplicate rows can be prevented. See the description of the RidgeMosaicCTMatch.sh script.

Table "overpass\_event"

Column | Type | Modifiers

------------------+--------------------------+------------------------

event\_num | integer | not null default nextval(

| | 'overpass\_event\_event\_num\_seq' )

sat\_id | character varying(15) |

orbit | integer | not null

radar\_id | character varying(15) |

overpass\_time | timestamp with time zone | not null

nearest\_distance | integer | not null

Indexes:

"overpass\_event\_pkey" primary key, btree (event\_num)

"overpass\_event\_sat\_id\_key" unique, btree (sat\_id, radar\_id, orbit)

Foreign-key constraints:

"$1" FOREIGN KEY (sat\_id) REFERENCES instrument(instrument\_id)

"$2" FOREIGN KEY (radar\_id) REFERENCES instrument(instrument\_id)

**rawradar table --** The ‘rawradar’ table contains the catalog of all original (not modified by quality control procedures) and default quality-controlled ground radar product files. *The ‘filename’ attribute itself is the Primary Key of the table, thus all raw radar products are assumed, and constrained, to have unique filenames by convention.* The ‘filepath’ is the partial path to the file, immediately preceding the filename, and following the common path to all raw ground radar files in the VN file system. See the identically-defined ‘gvradar’ table for field descriptions and constraints. ***This table is no longer used to catalog radar products in the GPM era, all radar products are now cataloged to the ‘gvradar’ table.***

Table "rawradar"

Column | Type | Modifiers

----------+--------------------------+-----------

product | character varying(15) |

radar\_id | character varying(15) |

nominal | timestamp with time zone |

filepath | character varying(63) |

filename | character varying(63) | not null

Indexes:

"rawradar\_pkey" primary key, btree (filename)

## Static Tables

Static tables hold information that does not vary case-by-case or rarely changes over time. These are the tables that define or constrain the dynamic data written by the VN data acquisition scripts. The static-data tables will be presented in alphabetical order by table name.

**fixed\_instrument\_location table –** The ‘fixed\_instrument\_location’ table defines the location information for fixed (non-mobile) observing sites – ground radars, typically. Sites must have a unique ‘instrument\_id’ value defined in the ‘instrument’ table before the location information can be added to this table. The ‘install\_date’ attribute is meant to be used in conjunction with the ‘replaced\_by\_id’ attribute in the ‘instrument’ table, to indicate when the switchover to the new ‘instrument\_id’ occurred, so that the proper ‘instrument\_id’ value can be associated to data for a given date, or to separate data at a given location by measuring instrument.

Table "fixed\_instrument\_location"

Column | Type | Modifiers

----------------+-----------------------+-----------

instrument\_id | character varying(15) | not null

install\_date | date | not null

state\_province | character(2) |

country | character(2) |

latitude | real | not null

longitude | real | not null

elevation | smallint |

Indexes:

"fixed\_instrument\_location\_pkey" primary key, btree (instrument\_id, install\_date)

Foreign-key constraints:

"$1" FOREIGN KEY (instrument\_id) REFERENCES instrument(instrument\_id)

**instrument table –** The ‘instrument’ table defines the unique identifiers (typically call letters of a station: KMLB, for instance) and descriptive attributes for observing platforms – ground radars and satellites, typically. Instruments must be defined in this table before data for the site can be cataloged in the database (see the ‘gvradar’, ‘orbit\_subset\_product’ and ‘productsubset’ tables, for instance). The parent\_child attribute identifies whether this instrument is bundled with other instruments (PR, TMI, LM) under a parent (the TRMM satellite) (parent\_child = ‘C’ [child] or ‘P’ [parent]) or stand-alone (like a NEXRAD radar) (parent\_child = ‘N’). Refer to the ‘lineage’ table for the definition of allowed ‘parent\_child’ values. The ‘produces\_data’ attribute indicates whether the instrument provides data products (parents typically don’t [e.g., TRMM] but children typically do [e.g., the PR]). The ‘fixed\_or\_moving’ attribute is self-explanatory. Those instruments marked as fixed (‘F’) will have their location information defined in the ‘fixed\_instrument\_location’ table, and those marked moving (‘M’) will have location information within the data itself. The ‘parent\_child’, ‘produces\_data’, and ‘fixed\_or\_moving’ attributes each have a default value that is assigned by the database if no value is specified when creating a new row in the ‘instrument’ table.

The ‘coverage\_type’ attribute textually describes how the instrument makes observations (horizontal scanning like a ground radar, cross-track scanning like the PR, or point observation like a surface weather station). The ‘replaced\_by\_id’ attribute gives the ‘instrument\_id’ of the instrument that replaced or took over observing responsibility for this instrument, if this is the case (station call letters may change over time, for instance).

Table "instrument"

Column | Type | Modifiers

-----------------+-----------------------+------------------------------

instrument\_id | character varying(15) | not null

instrument\_type | character varying(31) |

instrument\_name | character varying(31) |

owner | character varying(15) |

parent\_child | character(1) | not null default 'N'

produces\_data | character(1) | not null default 'Y'

fixed\_or\_moving | character(1) | not null default 'F'

coverage\_type | character varying(15) |

replaced\_by\_id | character varying(15) |

Indexes:

"instrument\_pkey" primary key, btree (instrument\_id)

Foreign-key constraints:

"$1" FOREIGN KEY (parent\_child) REFERENCES lineage(relationship)

**instrument\_hierarchy table –** When a parent/child relationship exists between instruments, the instrument\_hierarchy associates a parent instrument to its child instrument(s), as well as defining all the children belonging to a given parent. It resolves the one-to-many relationship between a parent instrument and its children. This table is not currently populated, but if it were, it would contain parent/child pairs like TRMM/PR, TRMM/TMI, GPM/GMI, GPM/DPR, GPM/Ka, GPM/Ku, METOPA/MHS, etc.

Table "instrument\_hierarchy"

Column | Type | Modifiers

----------------------+-----------------------+-----------

parent\_instrument\_id | character varying(15) | not null

child\_instrument\_id | character varying(15) | not null

Indexes:

"instrument\_hierarchy\_pkey" primary key, btree (parent\_instrument\_id, child\_instrument\_id)

Foreign-key constraints:

"$1" FOREIGN KEY (parent\_instrument\_id) REFERENCES instrument(instrument\_id)

"$2" FOREIGN KEY (child\_instrument\_id) REFERENCES instrument(instrument\_id)

**lineage table –** The ‘lineage’ table defines the allowed values for the parent-child relationship attribute associated with observing instruments: ‘C’ for Child, ‘P’ for Parent, or ‘N’ for None.

Table "lineage"

Column | Type | Modifiers

--------------+----------------------+-----------

relationship | character(1) | not null

description | character varying(6) | not null

Indexes:

"lineage\_pkey" primary key, btree (relationship)

**metadata\_parameter table –** The ‘metadata\_parameter’ table contains the plain-language description (‘parameter\_definition’ attribute) and PostGRESQL ‘data\_type’ associated with each ‘metadata\_id’ integer value. It defines which variable is being stored in a row of the ‘metadata\_temp’ and ‘event\_meta\_numeric’ tables.

Table "metadata\_parameter"

Column | Type | Modifiers

----------------------+-----------------------+-----------

metadata\_id | integer | not null

data\_type | character varying(10) | not null

parameter\_definition | character varying(63) | not null

Indexes:

"metadata\_parameter\_pkey" primary key, btree (metadata\_id)

**productsubset table –** The ‘productsubset’ table defines the orbital ‘subset’ identifiers that are associated with data products from a given satellite instrument (‘sat\_id’).

Table "productsubset"

Column | Type | Modifiers

--------+-----------------------+-----------

sat\_id | character varying(15) | not null

subset | character varying(15) | not null

Indexes:

"productsubset\_pkey" primary key, btree (sat\_id, subset)

Foreign-key constraints:

"instr\_fk" FOREIGN KEY (sat\_id) REFERENCES instrument(instrument\_id)

**siteproductsubset table –** The ‘siteproductsubset’ table, in conjunction with the ‘productsubset’ table, resolves the many-to-many relationship between orbital subsets and the radar sites within their area of coverage. For instance, the KMLB (Melbourne, FL) WSR-88D is associated with the ‘subGPMGV1’ area (for PR products produced for the core VN data area), the ‘CONUS’ area for GPM-era products, and the ‘MELB’ subset area (for legacy PR subset products). In general, for a given site, satellite, and orbit, only one of the product subsets will be present in the VN database and listed in the ‘orbit\_subset\_product’ table, but this is not enforced by the database or the VN software.

Table "siteproductsubset"

Column | Type | Modifiers

----------+-----------------------+-----------

sat\_id | character varying(15) | not null

subset | character varying(15) | not null

radar\_id | character varying(15) | not null

Indexes:

"siteproductsubset\_pkey" primary key, btree (sat\_id, subset, radar\_id)

Foreign-key constraints:

"instr\_fk" FOREIGN KEY (radar\_id) REFERENCES instrument(instrument\_id)

"subset\_fk" FOREIGN KEY (sat\_id, subset) REFERENCES productsubset(sat\_id, subset)

# Status Reporting and Data Archiving scripts

In addition to the data acquisition scripts described in Section 4, there are two additional scripts configured to run under the ‘gvoper’ crontab: archiveLogsMonthly.sh and notify\_vn\_status.sh. The archiveLogsMonthly.sh script is to move the daily log files for the VN processes into monthly ‘tar’ files to clean up the /data/log and /data/log/meta\_logs directories where they are created and would otherwise accumulate. The script is scheduled to run on the first day of each month, and the month’s log files to be archived is two months prior to the calendar month at the time the script runs, so at a minimum, the latest month’s log files are always left in place and unarchived.

The notify\_vn\_status.sh script produces a daily summary of VN data acquisition script status extracted from the log files for each cron-scheduled process, and sends the summary information as the text of an e-mail message to one or more recipients. It runs daily at 12:59 pm, after all the once-per-day data acquisition scripts are normally expected to have completed. It does not report on archiveLogsMonthly.sh.

# Backup scripts and procedures

Two types of data and code backups are performed for the GPM VN on the Linux data server ds1-gpmgv.gsfc.nasa.gov currently responsible for data collection, storage, and processing. The first backup is automated backup of the /data and /home directories to an external USB hard drive, and the second backup is a manual backup of the ‘gpmgv’ database on ds1-gpmgv to a local file using the pg\_dump utility of PostGRESQL. Database backups are done manually on an ad-hoc basis by running a command written for these purposes.

As of the GPM era, the /data/gpmgv data partition is mounted and shared between the ftp server hector.gsfc.nasa.gov and the VN data ingest host ds1-gpmgv.gsfc.nasa.gov. These shared data directories are backed up by default by the RAID5 arrangement of the multiple disk drives on the system. Data and user directories are also backed up to an external hard drive by the ds1-gpmgv systems administrator on a nightly basis. The ‘gpmgv’ database is supposed to be included in these latter backups as a result of their location within the directory structure, but this has not been tested.