Array Configuration Management Database

David DeBoer

February 26, 2019

Executive Summary
The primary scripts to view configuration management (see Sec. 3) data are:

- **parts.py**: various part/connection information is displayed as tables. Typing `parts.py info` will list additional information.
- **hookup.py**: system hookup information is shown. `hookup.py` will show all currently fully hooked up systems.
- **geo.py**: geographical information (primarily for antennas) is shown and may be plotted. `geo.py -g` will give a summary and graph.

1 Introduction

The HERA array/part configuration management database is a set of five tables within the larger hera_mc database, which is maintained on-site in the Karoo. The tables are detailed in the Appendix, but they are:

psql table	**in python file**	**with class name**
geo_location	hera_mc/geo_location.py	GeoLocation
station_type	hera_mc/geo_location.py	StationType
parts	hera_mc/part_connect.py	Parts
part_info	hera_mc/part_connect.py	PartInfo
connections	hera_mc/part_connect.py	Connections
cm_version	hera_mc/	
dubitable	hera_mc	

Software is contained in the repository https://github.com/HERA-Team/hera_mc.

The databases are structured primarily around *parts* and *connections*. *Parts* are meant to be single items that, in theory at least, are a thing than can be replaced as a unit. *Connections* define *ports* on a given parts and connect two ports together. Parts are all upper case and ports are all lower case.

All parts and connections are timed in that they have a start and stop time of operation. If stop is `None`, then it is active (it is given a date in the relatively far future). There are currently two special parts (one at each end of the signal chain) that are "geo_located" parts and have entries in the geo_location table: **station** and **node**.

Parts are hooked together via connections of their ports. The signal chain hook-up is given below and shown in Fig. 1.

- **Station**: geo_located position. See prefixes in table station_type (*e.g.* `HH` for herahex)
- **Antenna**: This is also the correlator number which now matches station number. **A**{`#`}
- **Feed**: element feed (Vivaldi). **FDV**{`#`}
- **FEM**: front-end module. **FEM**{`#`}
- **Cable-RFoF**: cable RF-over-fiber FEM-to-PAM. Number matches station/antenna. **CRF**{`#`}
- **PAM**: post-amp module in node. **PAM**{`#`}
- **SNAP**: digitizer/channelizer in node. **SNP**{`#`}

- **Node**: field-deployed node as a part **N**{#}
- **Node**: geo_located position for node as a station **ND**{#}

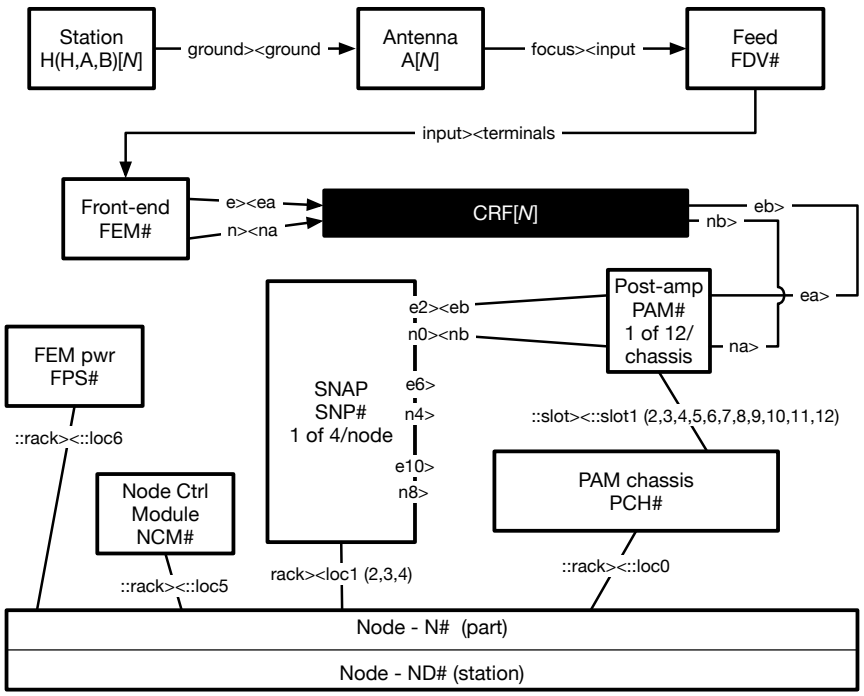

Figure 1: Block diagram of hookup. Filled boxes are cables, blue objects are in the receiverator and green objects are in the container. The line labels indicate the port names/connections.

2 Package Modules

This section provides a high-level overview of the python configuration management package modules within hera_mc that are called from scripts or used in interactive sessions.

geo_location.py	Defines `station_type` and `geo_location`. Provides update utility for `geo_location`. Contains `is_in_geo_location` and `is_in_connections`.
geo_handling.py	Contains myriad defs that handle various geo functionalities.
part_connect.py	Defines `parts_paper`, `part_info`, and `connections`. Provides update utilities for `parts_paper` and `connections`. Contains `__get_part_revisions`.
cm_table_info.py	Has mapping and order of tables and classes for initialization.
cm_handling.py	Defines the `Handling` class to handle various configuration management functionalities.
cm_transfer.py	Contains myriad defs that help package and initialize the cm tables.
cm_part_revisions.py	Contains myriad defs that deal with finding revision numbers etc.
cm_hookup.py	Defines the `Hookup` class to help determine and show the full part hookup.
cm_dataview.py	Contains various methods to display cm information.
cm_utils.py	Contains various defs called by other modules.

3 Scripts

This section provides a high-level overview of the high-level scripts: geo.py and parts.py. Note that info you type `info` after them, it presents information on them. The structure is

`geo.py` *action* `--arguments`

action is optional with a default.

3.1 geo.py

Has various plotting/printing options for station information. Available actions are (note you need the first 3 letters only):

- `geo` [default]: plots and/or lists (`-g`) antenna locations. Plots have the "background" behind them.
- `cofa`: plots and/or lists the position of the center of the array.
- `since`: plots and/or lists the antennas built since the provided date.
- `info`: provides information on the script.

`geo.py -g -l HH67` will show the "background" (selectable via –station-types or -t) of active (selectable via –show-state) antennas with HH67 highlighted. It will provide an overview printout as well.

3.2 parts.py

Has various printing options for part information (part info, connections, hookup, types, etc). Available actions are (note you only need the first 2 letters only):

- `hookup` [default]: lists hookup info for parts (as a csv list with no spaces) after -p
- `part_info`: provides a summary of given part(s)
- `conn_info`: provides a summary table of parts connected to given part(s)
- `rev_info`: provides a summary of revisions to given part(s)
- `types`: prints a summary table of part types
- `check_rev`: checks whether a given revision of a part exists
- `overlap_check`: checks that the given part doesn't have overlapping revisions.
- `info`: provides information on the script

`parts.py -p ri5` will provide a table of all hookups to receiverator 5. `parts.py -p ri5 --hookup-cols "station,cable-post-amp(in),f-engine"` will just show those columns.

4 Use Cases

Three use cases are identified:

1. In-the-field
2. Remote user
3. Bulk database

4.1 In-the-field updates

This is the "operational" mode, where inputs to the on-site database are done via scripts (command-line and/or web), which update the database directly. These scripts will be documented under the repository software.

4.2 Remote user

Remote use is the case when a remote user initializes and uses the tables outside of the main database. This assumes that postgresql etc is configured on the remote computer. Note that this mode supports both offline use (essentially steps 3-5 below, which are in bold) and development (probably all steps). The process is as follows:

1. Generate ascii files of the tables in the main database (in the Karoo) (`cm_pack.py [--tables table1,table2,...)`
2. Push to GitHub
3. **Pull down to remote computer**
4. **Install (`python setup.py install`)**
5. **Initialize the updated ones (`cm_init.py`)**
6. If you are developing, make a base version before you start (`cm_pack.py --base`)

Note that generally you should leave off the `--tables` option, since you should use all the tables so as to not break foreign keys (it is left there for "expert" users). Steps 3-5 are bolded, since if you just want to pull down the latest and use it assuming things are up-to-date they are all that are needed (i.e., leave the "packaging to the experts").

This has another option of `--base`, so that you can set up a base copy to which you can revert if need be by including `--base` on both commands (step 6).

4.3 Main database updates

This is effectively the opposite of the remote user case, bringing an updated set of table(s) back into the main database. The process is as follows:

1. Generate ascii files of the desired tables from the remote database: (`cm_pack.py --maindb $key [--tables table1,table2,...]`)
2. Push to GitHub
3. Pull down to site computer
4. Install
5. Update existing tables (`cm_init.py --maindb $key [--tables table1,table2,]`)

Note that `$key` is a user supplied string to just make sure you meant to do the main database update. It can be anything, you just need to remember it when you reload it on the main database computer qmaster. As per above, you should generally leave off the `--tables` option so as to do them all.

5 Tables

As mentioned above, there are five tables in the configuration management section of the database: (1) geo_location, (2) station_meta, (3) parts_paper, (4) part_info, (5) connections. The following tables summarize them with the following key:

- **Bold font** = primary key
- *Italics* = foreign_key.
- * = NotNull entries

Table 1: geo_location :: geo_location.GeoLocation

Column	Type	Description
station_name*	character varying(64)	Name of position - never changes
*station_type_name**	character varying(64)	Type of station
datum	character varying(64)	UTM datum
tile	character varying(64)	UTM tile
northing	double precision	UTM coordinate
easting	double precision	UTM coordinate
elevation	double precision	Elevation
created_gpstime*	BigInt	GPS second of creation.

Table 2: geo_location :: station_type.StationType

Column	Type	Description
station_type_name*	character varying(64)	Station type name
prefix*	character varying(64)	1-2 letter prefix for part station_name
description	character varying(64)	Short description
plot_marker	character varying(64)	Type of matplotlib marker

Table 3: part_connect :: parts_paper.Parts

Column	Type	Description
hpn*	character varying(64)	HERA part number
hpn_rev*	character varying(32)	HPN revision letter (A-Z)
hptype*	character varying(64)	HPN part type category
manufacturer_number	character varying(64)	Unique serial number for each part
start_gpstime*	BigInt	GPS second when part/rev is activated.
stop_gpstime	BigInt	GPS second when part/rev is de-activated

Table 4: part_connect :: part_info.PartInfo

Column	Type	Description
hpn*	character varying(64)	HERA part number
hpn_rev*	character varying(32)	HPN revision letter (A-Z)
posting_gpstime*	BigInt	GPS second information was posted
comment*	character varying(1024)	Comment
library_file	character varying(256)	Librarian filename (how to get it there?)

Table 5: part_connect :: connections.Connections

Column	Type	Description
upstream_part*	character varying(64)	Hera part number of upstream connection
up_part_rev*	character varying(32)	Hera part revision of upstream connection
upstream_output_port*	character varying(64)	Output port on upstream part
downstream_part*	character varying(64)	Hera part number of downstream connection
down_part_rev*	character varying(32)	Hera part revision of downstream connection
downstream_input_port*	character varying(64)	Input port on downstream part
start_gpstime*	BigInt	GPS second when connection started
stop_gpstime	BigInt	GPS second when connection ended