Web Services

Introduction

Everyone using MWA data needs access to metadata about the telescope itself, and about the observation/s that they are using. All of this metadata is stored in a PostgreSQL database on site, and this database is replicated at the Pawsey Centre in Perth, and also at MIT. Rather than access this data via SQL queries directly to the database (as these have the potential to place enormous load on the PostgreSQL server), the new access mechanism is via a set of 'web services'. Each of these services is a program that runs on a web server and accepts parameters in the URL, accesses the database to obtain and format some data, and returns it to the user.

The scripts in the 'MWA_Tools' repository has been updated to use these web services instead of using SQL queries, but anyone working with MWA data can also use these services directly if they prefer. Direct SQL access will soon be disabled.

The URL's for these web services can be entered directly into a web browser, used in shell scripts (with wget), or in your own code. Most of them return structures in JSON format - these are human readable strings, and JSON libraries are easy to find for most programming languages. Example Python code for grabbing data from these web services and JSON-decoding it are at the bottom of this page.

The URLs for these services all have a similar format, starting with the base URL, e.g.

> **Base Webservice URL**
>
> http://ws.mwatelescope.org/

> NOTE: The new URL above replaces the existing http://mwa-metadata01.pawsey.org.au address. Please update your code/scripts accordingly! Both work for now, but sometime in the future we will decommission the Pawsey URL.

There's an alternate base URL to use if your code is running on one of the computers on-site - contact Andrew Williams if you need it.

The base URL is followed by the service type (`metadata/` or `observation/`) and the service name (`find`, `obs`, `con`, `fits`, `tconv`, or `temps` for 'metadata' type services, or `errors`, `obs` or `skymap` for 'observation' type services). At the end of the URL are zero or more parameters (a question mark, followed by a series of name=value pairs separated by ampersands). The parameter values may need to be encoded if they contain characters not allowed in URL strings.

Web service descriptions and usage

Each of the web services is described below.

Find observations

Search the database for observations that satisfy given criteria.

For manual (human) use, omit all parameters and use the base URL to get a form where you can just fill in the relevant criteria: http://ws.mwatelescope.org/metadata/find

If you are writing code to search for observations, append the search criteria parameters to the URL.

eg, http://ws.mwatelescope.org/metadata/find?creator=randall&future=1

The full list of allowed constraints is:

- **mintime**, **maxtime**: Minimum and maximum values of 'starttime' for the observation. Times are INCLUSIVE of the endpoints, e.g. comparisons use 'starttime>=mintime and startttime<=maxtime'.
- **minduration**: Minimum observation duration (stoptime - starttime), in seconds.
- **minra**, **maxra**: Minimum and maximum values for ra_pointing for the observation, in degrees. NOTE - if **maxra** < **minra**, the values returned wrap around 0/360, e.g. use **maxra**=10 and **minra**=350.
- **mindec**, **maxdec**: Minimum and maximum values for dec_pointing for the observation, in degrees.
- **minel**, **maxel**: Minimum and maximum values for elevation_pointing for the observation, in degrees.
- **minaz**, **maxaz**: Minimum and maximum values for azimuth_pointing for the observation, in degrees. NOTE - if **maxaz** < **minaz**, the values returned wrap around 0/360, e.g. use **maxaz**=10 and **minaz**=350.
- **minlst**, **maxlst**: Minimum and maximum values for Local Sidereal Time for the observation, in degrees. NOTE - if **maxlst** < **minlst**, the values returned wrap around 0/360, e.g. use **maxlst**=10 and **minlst**=350.
- **minsunel**, **maxsunel**: Minimum and maximum values for sun_elevation for the observation, in degrees.
- **minsunpd**, **maxsunpd**: Minimum and maximum values for sun_pointing_distance for the observation, in degrees.
- **projectid**: A string to match against the projectid field for the observation.
- **mode**: A string to match against the observing mode, e.g. 'HW_LFILES', 'VOLTAGE_START', etc.
- **creator**: A string to match against the name of the person creating that observation.
- **obsname**: A string to match against the observation name.
- **contigfreq**: Pass `1` to match only observations with 24 contiguous frequency channels, or `0` to match only observations with gaps in the frequency channels.
- **calibration**: Pass `1` to match only calibration observations, or `0` to match only observations that are NOT calibrations.
- **cenchan**: Select only observations with this centre frequency channel.
- **anychan**: Select only observations that include this channel in their list of 24 frequency channels.
- **freq_res**: Select only observations with the correlator set to this frequency resolution (currently, the only valid numbers are 10, 20, and 40 kHz).
- **int_time**: Select only observations with the correlator set to this integration time (currently, the only valid numbers are 0.5, 1.0, 2.0 and 4.0 seconds).
- **future**: Pass `1` to match only observations that have not yet happened, or `0` to match only observations that are in the past.
- **dataquality**: Pass one or more possible data quality flags (digits from `1` to `6`) to match against. The values are `1` (Good), `2` (Some Issues), `3` (U nusable), `4` (Deleted), `5` (Marked for deletion) or `6` (Processed). For example, passing `1` returns only 'Good' observations, `12`returns only 'Good' or 'Some Issues', `345` returns only unusable/deleted/marked-for-deletion observations, etc.
- **gridpoint**: An integer to match against the gridpoint_number column in the schedule_metadata table.
- **minfiles**: Select only observations with at least this many data files recorded.

Other parameters control the output, and output format:

- **extended**: If specified, an extended set of parameters are returned in the output list (see below).
- **dict**: If specified, a dictionary is returned, instead of a list, with even more parameters (see below).
- **pagesize**: By default, the web service returns a maximum of 200 observations at a time. Use this parameter to specify a smaller page size, or (with care) a larger one.
- **page**: By default, the web service returns the first **pagesize** results. If you pass a number in the `page` argument, then that 'page' of results is returned instead.
- **html**: If this parameter is given, the results are returned in human-readable format as a web page, instead of a JSON structure. The SQL query that generated the result will be shown at the top, and if there are more than **pagesize** observations, buttons to skip to other pages will be shown as well. Clicking on the obsid in the results takes you to the observation summary page (described below).

Any constraints that take strings (**projectid**, **mode**, **obsname** and **creator**) accept the standard PostgreSQL wildcard characters. A percent character (`%`) means 'match zero or more of any character', and an underscore (_) means 'match exactly one of any character'. If you wish to match an actual percent or underscore character, escape it with a single leading backslash (e.g. `foo\_bar`).

Normally, the parameters returned are:

- starttime (obsid in GPS seconds)
- obsname
- creator
- projectid
- ra_pointing (in degrees)
- dec_pointing (in degrees)

if you pass the the **extended** parameter, then more parameters are returned for each matching observation:

- starttime (obsid in GPS seconds)
- stoptime (in GPS seconds)
- obsname
- creator
- projectid
- ra_pointing (in degrees)
- dec_pointing (in degrees)
- azimuth (in degrees)
- elevation (in degrees)
- ra_phase_center (in degrees)
- dec_phase_center (in degrees)
- Local sidereal time (in degrees)
- gridpoint number

If you pass the **dict** parameter, then a dictionary is returned, containing even more parameters:

- starttime (obsid in GPS seconds)
- stoptime (in GPS seconds)
- obsname

- creator
- projectid
- ra_pointing (in degrees)
- dec_pointing (in degrees)
- azimuth (in degrees)
- elevation (in degrees)
- ra_phase_center (in degrees)
- dec_phase_center (in degrees)
- Local sidereal time (in degrees)
- gridpoint name ('EOR1' or 'sweet')
- gridpoint number
- dataquality
- dataqualitycomment
- mode (eg 'HW_LFILES')
- int_time (correlator integration time)
- freq_res (correlator frequency resolution)
- frequencies (list of 24 coarse channel numbers)

If you pass `1` to the **`pretty`** parameter, than the JSON results are indented to make them more human readable (the default when a GUI web browser is used to make the query). If `0` is passed, then the results are not indented.

Get observation/schedule data for an observation

eg, http://ws.mwatelescope.org/metadata/obs?obs_id=1096952256 or http://ws.mwatelescope.org/metadata/obs?filename=1095675784_20140925102450_gpubox23_01.fits

This service returns details about a single observation. The observation can be specified by passing either an observation ID (starttime in GPS seconds) with the **`obs_id`** or **`obsid`** parameter, or by passing the full name of a data file recorded by that observation using the **`filename`** parameter. If neither **`obs_id`**/**`obsid`** or **`filename`** are passed, data is returned for the most recent observation actually observed by the telescope.

If the **`filetype`** parameter is passed, only files of that type are returned in the observation info structure (e.g. **`filetype`**=8 gives you normal correlated data files, **`filetype`**=1 gives you flag files). This parameter is only likely to be useful for voltage capture observations, which can contain thousands of files - you can use the **`filetype`** argument to have some (or none) of these file details returned, to speed up the query.

If an observation taken more than one week ago is queried, the details are cached on the server, to speed up subsequent queries about the same object. Observations in the future, or less than a few days old, will have details in the database change as tiles are flagged or observation logs are analysed, so it's not worth saving the data in the query cache.

If you are making a query, and you **don't** want the results taken from the cache (you want them to be read directly from the database) than add the **`nocach`** parameter to your query. Note that this can be hundreds of times slower than using the cached data, so this parameter should be used rarely.

If you pass `1` to the **`pretty`** parameter, than the JSON results are indented to make them more human readable (the default when a GUI web browser is used to make the query). If `0` is passed, then the results are not indented.

This service returns a hierarchical structure (as a string in JSON format) that contains every detail about that observation. That string can then be converted from JSON format to a Python dictionary - an example would be:


```
{
    "starttime": 1096952256
    "stoptime": 1096952552,
    "freq_res": 40,
    "int_time": 0.5,
    "dataquality": 1,
    "dataqualitycomment": "",
    "dataready": true,
    "origkey": [1096952256],
    "log": null,
    "dec_phase_center": -6.5297643697349796,
    "ra_phase_center": 194.69351514167101,
    "unpowered_tile_name": "default",
    "new": false,
    "voltagebeams": [],
    "obsname": "Sun",
    "rtssettings": null,
    "mode": "HW_LFILES",
    "creator": "rwayth",
    "projectid": "G0002",
```

```
    "faults": {
        "badgains": {},
        "rawerrors": ["No status for rec15"],
        "cookederrors": ["Bad Receiver State:[rec15]"],
        "badpointings": {},
        "badfreqs": {},
        "flagged": {},
        "badstates": {"15": [151, 152, 153, 154, 155, 156, 157, 158]
        }
    },

    "bad_tiles": [151, 152, 153, 154, 155, 156, 157, 158],

    "logs": {
        "eab54fbe-4fa8-11e4-ac8a-001e689b5818": {
            "comment": "/usr/local/bin/single_observation.py --starttime=2014-10-10,04:57:18 --stoptime=++296s
--freq=62;63;69;70;76;77;84;85;93;94;103;104;113;114;125;126;139;140;153;154;169;170;187;188 --obsname=Sun --
inttime=0.5 --freqres=40 --creator=rwayth --useazel --usegrid= --project=G0002 --gain_control_value=14 --
source=Sun",
            "modtime": "2014-10-09T11:39:23.876102+00:00",
            "observation": null,
            "creator": "rwayth",
            "log_id": "eab54fbe-4fa8-11e4-ac8a-001e689b5818",
            "referencetime": 1096952256.0,
            "logtype": 5,
            "endtime": 1096952552.0
        },
        "222b3506-50e6-11e4-844c-001e689b5818": {
            "comment": "No status for rec15",
            "modtime": "2014-10-11T01:30:07.390286+00:00",
            "observation": null,
            "creator": "ocheck.py",
            "log_id": "222b3506-50e6-11e4-844c-001e689b5818",
            "referencetime": 1096952256.0,
            "logtype": 7,
            "endtime": 1096952552.0
        }

"alldelays": {
        "11": [  [15, 13, 11, 9, 12, 10, 8, 6, 9, 7, 5, 3, 6, 4, 2, 0], [15, 13, 11, 9, 12, 10, 8, 6, 9, 7, 5,
3, 6, 4, 2, 0]  ],

        "12": [  [15, 13, 11, 9, 12, 10, 8, 6, 9, 7, 5, 3, 6, 4, 2, 0], [15, 13, 11, 9, 12, 10, 8, 6, 9, 7, 5,
3, 6, 4, 2, 0]  ],

       …

        "167": [  [15, 13, 11, 9, 12, 10, 8, 6, 9, 7, 5, 3, 6, 4, 2, 0], [15, 13, 11, 9, 12, 10, 8, 6, 9, 7,
5, 3, 6, 4, 2, 0]  ],

        "168": [  [15, 13, 11, 9, 12, 10, 8, 6, 9, 7, 5, 3, 6, 4, 2, 0], [15, 13, 11, 9, 12, 10, 8, 6, 9, 32,
5, 3, 6, 4, 2, 0]  ]

       }

"bftemps": {
        "11": ["Tile011", 1, 1, 49.29999999999998],
        "12": ["Tile012", 1, 2, 48.399999999999984],

       …

       "167": ["Tile167", 16, 7, 47.70000000000002],

        "168": ["Tile168", 16, 8, 46.899999999999984]
        },

    "rfstreams": {
        "0": {
            "starttime": 1096952256,
            "number": 0,
            "creator": "rwayth",
            "bad_tiles": [151,152,153,154,155,156,157,158],
            "origkey": [1096952256, 0],
            "delays": [15, 13, 11, 9, 12, 10, 8, 6, 9, 7, 5, 3, 6, 4, 2, 0],
            "hex": "",
            "ra": null,
            "frequency_type": "CHANNEL",
            "new": false,
            "gain_control_value": 14.0,
            "elevation": 64.693399999999997,
            "frequencies": [62, 63, 69, 70, 76, 77, 84, 85, 93, 94, 103, 104, 113, 114, 125, 126, 139, 140,
153, 154, 169, 170, 187, 188],
            "gain_control_type": "DB_BF",
```

```
            "dipole_exclusion": "default",
            "walsh_mode": "OFF",
            "vsib_frequency": 113,
            "azimuth": 326.31,
            "dec": null,
            "tile_selection": "all_on"

            "bad_dipoles": {
                "153": [ [16], [13] ],
                "155": [ [16], [] ],
                "157": [ [16], [] ],
                "156": [ [], [13] ],
                "114": [ [12], [] ],
                "131": [ [], [15] ],
                "111": [ [10], [] ],
                "136": [ [16], [16] ],
                "113": [ [6], [] ],
                "68": [ [13], [] ],
                "83": [ [], [9] ],
                "138": [ [13], [] ],
                "87": [ [], [10] ],
                "26": [ [], [16] ],
                "22": [ [], [13] ],
                "43": [ [14], [] ],
                "122": [ [], [1] ],
                "143": [ [], [13] ],
                "141": [ [8], [] ],
                "77": [ [13], [] ],
                "121": [ [14], [] ],
                "75": [ [1], [] ],
                "168": [ [], [10] ],
                "126": [ [], [9] ],
                "92": [ [1], [] ],
                "95": [ [6], [] ],
                "107": [ [9], [] ],
                "97": [  [], [15] ],
                "163": [ [], [13] ],
                "13": [ [14], [1] ],
                "17": [ [13], [] ],
                "31": [ [5], [] ],
                "88": [ [16], [] ],
                "34": [ [10], [] ],
                "73": [ [14], [] ]  },

            "tileset": {
                "ylist": [11, 12, 13, 14, 15, 16, 17, 18, 21, 22, 23, 24, 25, 26, 27, 28, 31, 32, 33, 34, 35,
36, 37, 38,
41, 42, 43, 44, 45, 46, 47, 48, 51, 52, 53, 54, 55, 56, 57, 58, 61, 62, 63, 64, 65, 66, 67, 68, 71, 72, 73,
74, 75, 76, 77, 78,
81, 82, 83, 84, 85, 86, 87, 88, 91, 92, 93, 94, 95, 96, 97, 98, 101, 102, 103, 104, 105, 106, 107, 108, 111,
112, 113, 114,
115, 116, 117, 118, 121, 122, 123, 124, 125, 126, 127, 128, 131, 132, 133, 134, 135, 136, 137, 138, 141, 142,
143, 144,
145, 146, 147, 148, 151, 152, 153, 154, 155, 156, 157, 158, 161, 162, 163, 164, 165, 166, 167, 168],
                "xlist": [11, 12, 13, 14, 15, 16, 17, 18, 21, 22, 23, 24, 25, 26, 27, 28, 31, 32, 33, 34, 35,
36, 37, 38,
41, 42, 43, 44, 45, 46, 47, 48, 51, 52, 53, 54, 55, 56, 57, 58, 61, 62, 63, 64, 65, 66, 67, 68, 71, 72, 73,
74, 75, 76, 77, 78,
81, 82, 83, 84, 85, 86, 87, 88, 91, 92, 93, 94, 95, 96, 97, 98, 101, 102, 103, 104, 105, 106, 107, 108, 111,
112, 113, 114,
115, 116, 117, 118, 121, 122, 123, 124, 125, 126, 127, 128, 131, 132, 133, 134, 135, 136, 137, 138, 141, 142,
143, 144,
145, 146, 147, 148, 151, 152, 153, 154, 155, 156, 157, 158, 161, 162, 163, 164, 165, 166, 167, 168],
                "name": "all_on",
                "creator": "mwa"
            },

        }
    },
    "metadata": {
        "local_sidereal_time_deg": 209.72987912278401,
        "ra_pointing": 195.77112036706501,
        "elevation_pointing": 64.693399999999997,
        "calibration": false,
        "azimuth_pointing": 326.30990000000003,
        "gridpoint_name": "sweet",
        "jupiter_pointing_distance": 59.199361313084701,
        "sun_elevation": 65.766122499057701,
        "sky_temp": 378.79364146732502,
        "dec_pointing": -5.0010847948659203,
        "moon_pointing_distance": 156.59922736056001,
        "sun_pointing_distance": 1.5955869339600499,
        "gridpoint_number": 44,
        "calibrators": ""
```

```
    },
    "files": {
        "1096952256_20141010045822_gpubox20_01.fits": {
            "site_path": "http://mwangas/RETRIEVE?file_id=1096952256_20141010045822_gpubox20_01.fits",
            "host": "gpubox20",
            "filetype": 8,
            "remote_archived": true,
            "deleted": false,
            "size": 1015030080
        },

        ...

        "1096952256_20141010045922_gpubox04_02.fits": {
            "site_path": "http://mwangas/RETRIEVE?file_id=1096952256_20141010045922_gpubox04_02.fits",
            "host": "gpubox04",
            "filetype": 8,
            "remote_archived": true,
            "deleted": false,
            "size": 1015030080
        }

    "tdict": {
        "11": ["Tile011", 1, 1, false],
        "12": ["Tile012", 1, 2, false],

       ...

        "167": ["Tile167", 16, 7, false],
        "168": ["Tile168", 16, 8, false]
        },

    },
}
```


Note that this hierarchical structure is almost identical to the structure returned by schedule.MWA_Setting() - the main difference is that any values uses as keys to dictionaries (e.g. the 'number' in the .rfstreams dictionary, or the tile ID in the bad dipoles structure) is a string representing a decimal number, instead of an integer value.

Get telescope configuration for a given observation

eg, http://ws.mwatelescope.org/metadata/con?obs_id=1096952256 or http://ws.mwatelescope.org/metadata/con?filename=1095675784_20140925102450_gpubox23_01.fits

This service takes an observation ID, in gpsseconds, and returns a hierarchical structure (as a string in JSON format) that contains every detail about the configuration of the array at the time of that observation (the value passed can actually be any time in gpsseconds, it doesn't have to be the start time of an actual observation). The time can be passed as **`obs_id`**, **`obsid`** or **`reftime`** parameter, all are equivalent. If you only want a summary of the configuration (to know whether the array is in 'PHASE1', 'COMPACT' or 'LB' configuration) then pass the '**summary**' parameter in the URL.

If an observation ID more than one week in the past is queried, the details are cached on the server, to speed up subsequent queries about the same object. Observations in the future, or less than a few days old, will have details in the database change as tiles are flagged or observation logs are analysed, so it's not worth saving the data in the query cache.

If you pass `1` to the **`pretty`** parameter, than the JSON results are indented to make them more human readable (the default when a GUI web browser is used to make the query). If `0` is passed, then the results are not indented.

If you are making a query, and you **don't** want the results taken from the cache (you want them to be read directly from the database) than add the **`nocach`** parameter to your query. Note that this can be hundreds of times slower than using the cached data, so this parameter should be used rarely.


```
{
```

```
        "11": {
            "id": 11
            "receiver": 1,
            "slot": 1,
            "bf": 11,
            "flagged": null,
            "bfgainx": 6.0,
            "dgains": [
    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,367,362,354,339,312
    ,270,
    219,173,140,123,115,108,100,91,83,78,76,78,80,82,82,81,80,79,80,82,85,88,89,89,88,87,86,87,89,91,92,92,92,91,89
    ,89,88,89,90,91,
    91,90,89,87,86,85,85,85,85,85,85,84,83,81,80,78,78,77,77,77,77,76,74,73,71,70,69,69,69,68,68,67,66,65,64,64,64,
    64,64,65,65,65,
    64,64,64,65,65,66,67,67,68,68,69,70,71,72,73,74,75,76,77,77,78,79,80,82,83,84,85,86,87,88,89,90,92,93,95,97,99,
    100,102,104,106,
    108,110,113,116,119,121,124,127,130,133,136,140,145,149,154,158,161,165,168,172,175,179,182,186,189,193,196,200
    ,204,207,
    211,214,218,221,225,228,232,235,238,240,241,243,245,247,250,253,255,258,260,261,262,262,263,264,267,270,274,278
    ,282,285,
    289,293,298,304,311,318,325,332,338,343,348,352,355,358,361,363,365,366,367,368,368
            ],
            "catten": 18.0,
            "ted": -656.139138187221,
            "bfgainy": 6.0,
            "pos": [
                -149.785,
                265.814
            ],
            "inputnum": 44,
            "altitude": 377.011,
            "flavor": "RG6_90",
        },

        "12": {
            "id": 12
            "receiver": 1,
            "slot": 2,
            "bf": 12,
            "flagged": null,
            "bfgainx": 6.0,

    ...

    }
```

The structure is a dictionary, with tile ID as a string and an entry for every tile. Each entry is another dictionary, containing data about that tile - the connectivity (receiver number, receiver slot, and correlator 'inputnum'), location ('pos' is a tuple containing east/north in metres, 'alt' is the altitude in metres), and cable data (digital gain multipliers, total tile electrical delay ('ted'), cable and beamformer attenuation).

Get a 'metafits' file with observation AND telescope configuration data, for a given observation

eg, http://ws.mwatelescope.org/metadata/fits?obs_id=1096952256 or http://ws.mwatelescope.org/metadata/fits?filename=1095675784_20140925102450_gpubox23_01.fits

This service takes returns a metafits file for a given observation. The observation can be specified by passing either an observation ID (starttime in GPS seconds) with the '**obs_id**' parameter, or by passing the full name of a data file recorded by that observation using the '**filename**' parameter. If neither **obs_id** or **filename** are passed, a metafits file is returned for the most recent observation actually observed by the telescope.

There are two optional parameters that affect the contents of the metafits file:

- **`lock`**: If **lock**=1, write 'HA_HRS' (hour angle at the start of the observation) in the header. If **lock**=0 (the default), write 'RA_HRS' (RA of the desired phase centre) in the header.
- **`min_bad_dipoles`**: A number from 1-16. If either polarisation on a single tile has this many dipoles (or more), flag it as bad. The default (**min_bad_dipoles**=2) means that if a tile has more than one bad dipole in either or both of X and Y, it will be flagged.
- **nocache**: Don't use cached observation data.

If you are making a query, and you **don't** want the results taken from the cache (you want them to be read directly from the database) than add the **nocache** parameter to your query. Note that this can be hundreds of times slower than using the cached data, so this parameter should be used rarely.

Get the best calibration solution available for a target observation ID (experimental!)

e.g. http://ws.mwatelescope.org/calib/get_calfile_for_obsid?obs_id=1120493008

Where:

- obs_id is an observation ID you wish to calibrate. get_calfile_for_obsid will search the calibration solution database for the best calibration solution available for this observation. See MWA ASVO: Calibration Option for some background information on the calibration database and how it is populated.

This service will return:

- On Success:
  - A zip file containing:
    - cal_id.bin: Binary file (in AO-cal format) containing the calibration solution. This can be used in Andre Offringa's calibrate and Cotter tools. Note the filename contains the observation ID of the calibrator observation used.
    - obs_id_flagged_tiles.txt: A plain text file listing the tile_id's of tiles that failed to calibrate.
    - obs_id_flags.py: A simple python script which you can run to apply the calibration solution (in CASA).
    - obs_id.metafits: A metafits file, with the calibration solution applied to the data.
- On Error:
  - http status 400 if the cal_id is not found in the calibration solution database.
  - http status 500 if an internal error occurred.

Note that this service will only look for calibration observations taken with the same analogue attenuation ('gain_control_value') as the observation. If you wish to search for /any/ calibration solution, regardless of analogue attenuation, add the 'any_gain' parameter to the URL string - for example, http://ws.mwatelescope.org/calib/get_calfile_for_obsid?obs_id=1120493008&any_gain

Get the calibration solution for a specific calibrator observation ID (experimental!)

e.g. http://ws.mwatelescope.org/calib/get_calfile_for_calid?cal_id=1120493424

Where:

- cal_id is the observation ID of an observation that has been marked as a calibrator observation.

This service will return:

- On Success:
  - A zip file containing:
    - cal_id.bin: Binary file (in AO-cal format) containing the calibration solution. This can be used in Andre Offringa's calibrate and Cotter tools.
    - cal_id_flagged_tiles.txt: A plain text file listing the tile_id's of tiles that failed to calibrate.
    - cal_id_flags.py: A simple python script which you can run to apply the calibration solution (in CASA).
- On Error:
  - http status 400 if the cal_id is not found in the calibration solution database.
  - http status 500 if an internal error occurred.

Convert times/dates between UTC strings and GPS seconds values

eg, http://ws.mwatelescope.org/metadata/tconv?gpssec=1096952256 or http://ws.mwatelescope.org/metadata/tconv/?utciso=2014-10-10T04:57:20

This service translates between UTC times (as an ISO format string) and times in 'GPS seconds' - seconds since the GPS epoch, without any leap seconds. The parameters are:

- **`gpssec`**: An **integer** number, the time in GPS seconds, e.g. 1096952256. An ISO format time string is returned.
- **`utciso`**: An ISO format date/time string in UTC - e.g. "2014-10-10T04:57:20". The 'T' separating the date and time parts can be replaced by a space character, but if so it must be URL encoded ('%20').

Get beamformer temperatures for a given observation ID

e.g. http://ws.mwatelescope.org/metadata/temps/?starttime=1174096840

This service takes a time in gpsseconds (typically an observation ID) and returns a structure indexed by the tile ID, giving the tile name, the receiver ID (1-16), the slot ID (1-8) and the beamformer temperature in degrees C for that beamformer at the given time (actually the average of all measurements over the next 5 minutes after the given time).

Note that the beamformer temperatures are only actually read from the beamformer when a pointing command is sent (at the start of each observation), so if you specify a time in between observations, the vlaues will be stale - possibly many hours old. A temperature value of 'None' indicates communications errors with the tile, showing that no reading was returned.

The parameters are:

- **`starttime`**: An integer time in GPS seconds, typically the starttime of an observation.

If you pass `1` to the **`pretty`** parameter, than the JSON results are indented to make them more human readable (the default when a GUI web browser is used to make the query). If `0` is passed, then the results are not indented.

Delete the server-side cache entries associated with a given observation ID

eg, http://ws.mwatelescope.org/metadata/invalcache?obs_id=1096952256 or http://ws.mwatelescope.org/metadata/invalcache?filename=1095675784_20140925102450_gpubox23_01.fits

Deletes the data cached on the web server about a given observation, so that the next query will re-load the data from the SQL database. The observation can be specified by passing either an observation ID (starttime in GPS seconds) with the **obs_id** or **obsid** parameter, or by passing the full name of a data file recorded by that observation using the **filename** parameter.

Get all of the data files associated with a given observation ID

eg, http://ws.mwatelescope.org/metadata/data_files?obs_id=1096952256 or http://ws.mwatelescope.org/metadata/data_files?filename=1095675784_20140925102450_gpubox23_01.fits

This service returns details about a single observation. The observation can be specified by passing either an observation ID (starttime in GPS seconds) with the **obs_id** or **obsid** parameter, or by passing the full name of a data file recorded by that observation using the **filename** parameter.

The parameters are:

- **obs_id**: An observation ID (GPS start time in seconds)
- **filename**: One data file name associated with anobservation, instead of an obs_id
- **mintime**: Minimum data file timestamp, as an integer GPS second value. Only works for VCS observations.
- **maxtime**: Maximum data file timestamp, as an integer GPS second value. Only works for VCS observations.
- **pretty**: `1` to have the JSON results are indented to make them more human readable (the default when a GUI web browser is used to make the query). If `0` is passed (the default), then the results are not indented.

Note that mintime is inclusive (>=), while maxtime is not inclusive (<), like Python ranges.

Example output is:


```
{
    "1096952256_20141010045822_gpubox20_01.fits": {
        "site_path": "http://mwangas/RETRIEVE?file_id=1096952256_20141010045822_gpubox20_01.fits",
        "remote_archived": true,
        "host": "gpubox20",
        "deleted": false,
        "filetype": 8,
        "size": 1015030080
    },
    "1096952256_20141010050122_gpubox07_04.fits": {
        "site_path": "http://mwangas/RETRIEVE?file_id=1096952256_20141010050122_gpubox07_04.fits",
        "remote_archived": true,
        "host": "gpubox07",
        "deleted": false,
        "filetype": 8,
        "size": 896610240
    },
    "1096952256_20141010050022_gpubox11_03.fits": {
        "site_path": "http://mwangas/RETRIEVE?file_id=1096952256_20141010050022_gpubox11_03.fits",
        "remote_archived": true,
        "host": "gpubox11",
        "deleted": false,
        "filetype": 8,
        "size": 1015030080
    },

...


    "1096952256_20141010045922_gpubox04_02.fits": {
        "site_path": "http://mwangas/RETRIEVE?file_id=1096952256_20141010045922_gpubox04_02.fits",
        "remote_archived": true,
        "host": "gpubox04",
        "deleted": false,
        "filetype": 8,
        "size": 1015030080
    }
}
```


Get a summary of hardware/software errors that affected a given observation

e.g. http://ws.mwatelescope.org/observation/errors/?obsid=1174096840

This service takes an observation ID and returns a summary of the errors (failures to communicate with receivers and/or beamformers) that affected that observation. By default, the service will return a colour-coded HTML table, but several forms of machine readable (JSON) output are available as well.

The parameters are:

- **obsid** or **obs_id**: The starttime of an observation in GPS seconds. If not supplied, the most recent observation is used.
- **filename**: If **obsid** is not given, you can specify the name of a data file generated by the given observation instead.

- **`maxbaddipoles`**: By default, a tile will be marked as having a bad beamshape if more than one bad dipole is present in the same polarisation on that tile. This parameter allows the user to specify a different threshold.
- **`refresh`**: If this parameter is included, the HTML table returned will include a REFRESH tag, to force the browser to update it every 60 seconds.
- **`flip`**: If this parameter is included, the slot order (8-1 from top to bottom) is flipped, to show slot 1 tiles at the top of the table.
- **`raw`**: Return a JSON structure containing a list of human-readable strings summarising the receiver/tile communication errors.
- **`cooked`**: Return a JSON structure containing a list of human-readable strings summarising the errors in a different format.
- **`dictformat`**: Return a JSON structure containing a fully machine readable list of tile IDs with problems in each of the possible categories (badpointings, badfreqs, badgains, badstates, badbeamshape and flagged), as well as the cooked and raw format lists of strings.

If you pass `1` to the **`pretty`** parameter, than the JSON results are indented to make them more human readable (the default when a GUI web browser is used to make the query). If `0` is passed, then the results are not indented.

The dictformat output contains:

- **badstates**: a dict of all recid/tileids where the receiver wasn't observing at all
- **badpointings**: a dict of all recid/tileids that had bad comms to the beamformer to set the delays
- **badfreqs**: a dict of all recid/tileids where the ADFB wasn't set to the correct coarse channel set
- **badgains**: a dict of all recid/tileids where the analogue attenuation wasn't set correctly
- **badbeamshape**: a dict of all recid/tileids where there are two or more bad dipoles in the same polarisation
- **flagged**: a dict of all recid/tileids where we have manually flagged the tile as bad
- **tileflags**: All of the tiles in that observation that were manually flagged as bad, and when/why
- **rawerrors**, **cookederrors**, **shortstring**, **htmlstring**: human-readable strings to summarise those errors.

An example of dictformat output is:


```
{
    "badgains": {},
    "rawerrors": [
        "No status for rec05",
        "Rec03:",
        "    Mismatch in xdelaysetting[4] - Az/El=(None,None) should be Az/El=(206.6,74.6)",
        "    Mismatch in ydelaysetting[4] - Az/El=(None,None) should be Az/El=(206.6,74.6)",
        "Rec07:",
        "    Mismatch in xdelaysetting[3] - Az/El=(90.0,76.3) should be Az/El=(206.6,74.6)",
        "    Mismatch in ydelaysetting[3] - Az/El=(90.0,76.3) should be Az/El=(206.6,74.6)",
        "Receivers Rec12 - Rec13",
        "    Mismatch in xdelaysetting[8] - Az/El=(None,None) should be Az/El=(206.6,74.6)",
        "    Mismatch in ydelaysetting[8] - Az/El=(None,None) should be Az/El=(206.6,74.6)",
        "Rec15:",
        "    Mismatch in xdelaysetting[7] - Az/El=(None,None) should be Az/El=(77.8,68.7)",
        "    Mismatch in ydelaysetting[7] - Az/El=(None,None) should be Az/El=(206.6,74.6)",
        "PyController failed to point tile/s: [LBA3]"
    ],
    "cookederrors": [
        "Bad Receiver State:[rec05]",
        "Bad pointing:\n       rec01:[LBA3]\n       rec03:[LBF4]\n       rec07:[Tile073]\n       rec12:
[Tile128]\n       rec13:[LBE8]\n       rec15:[Tile157]",
        "Bad Beam Shape:\n       rec06:[Tile142,Tile144]\n       rec07:[Tile072,Tile074,Tile078]\n       rec10:
[Tile103,Tile104]\n       rec11:[Tile117]\n       rec12:[Tile123,Tile127]\n       rec14:[LBC4,LBC7]\n
rec15:[Tile155,Tile156]\n       rec16:[Tile161,Tile167]"
    ],
    "badpointings": {
        "1": [2003],
        "3": [2044],
        "7": [73],
        "12": [128],
        "13": [2040],
        "15": [157]
    },
    "badstates": {
        "5": [51, 52, 53, 54, 55, 56, 57, 58]
    },
    "badbeamshape": {
        "6": [142, 144],
        "7": [72, 74, 78],
        "10": [103, 104],
        "11": [117],
        "12": [123, 127],
        "14": [2020, 2023],
        "15": [155, 156],
        "16": [161, 167]
    },
    "flagged": {
        "8": [2016],
        "13": [2040],
        "14": [2020, 2022, 2024]
    },
    "badfreqs": {}
}
```

Get a sky map with the MWA beam shape for an observation

e.g. http://ws.mwatelescope.org/observation/skymap/?obsid=1174096840

This service takes an observation ID and returns a sky map showing the MWA beam for an observation, superimposed on a HASLAM background image with radio sources.

The parameters are:

- **`obsid`** or **`obs_id`**: The starttime of an observation in GPS seconds. If not supplied, the most recent observation is used.
- **`filename`**: If **obsid** is not given, you can specify the name of a data file generated by the given observation instead.
- **`plotscale`**: If not specified, a 1200x1200 pixel PNG is returned. If **plotscale**=0.5 (for example), a 600x600 pixel image is returned instead.
- **`hidenulls`**: By default, the beam nulls are shown by including (black) contour lines at power levels of 0.001. If this parameter is passed, the null contour lines are not drawn.
- **`channel`**: By default, the centre frequency for the observation is used for the beam model. If this parameter is given, this MWA channel number (0-255) is used to generate the beam shape instead.

Note that this service returns binary data containing a PNG image, not a JSON structure. Use the URL in a web browser, use 'wget' or 'curl' from the command line and save to 'filename.png', or use the urllib2.urlopen() function, and call the read() method on the connection object that it returns to read the binary data.

Get a human-readable observation summary page

e.g. http://ws.mwatelescope.org/observation/obs/?obsid=1174096840

This service takes an observation ID and returns a human readable web page summarising the observation, including any errors, files recorded, etc.

The parameters are:

- **`obsid`** or **`obs_id`**: The starttime of an observation in GPS seconds. If not supplied, the most recent observation is used.
- **`filename`**: If **obsid** is not given, you can specify the name of a data file generated by the given observation instead.
- **`hidenulls`**: By default, the sky map beam nulls are shown by including (black) contour lines at power levels of 0.001. If this parameter is passed, the null contour lines are not drawn.
- **`flip`**: If this parameter is included, the slot order in the colour coded error table (8-1 from top to bottom) is flipped, to show slot 1 tiles at the top of the table.
- **`maxbaddipoles`**: By default, a tile will be marked as having a bad beamshape if more than one bad dipole is present in the same polarisation on that tile. This parameter allows the user to specify a different threshold.
- **`nocache`**: Don't use cached observation data.

If you are making a query, and you **don't** want the results taken from the cache (you want them to be read directly from the database) than add the **`nocach`** parameter to your query.

Example Python code to use these web services

```
#### Python 2.X code ################################################
import urllib
import urllib2
import json

# Append the service name to this base URL, e.g. 'con', 'obs', etc.
BASEURL = 'http://ws.mwatelescope.org/'

# Function to call a JSON web service and return a dictionary:

def getmeta(servicetype='metadata', service='obs', params=None):
    """Given a JSON web servicetype ('observation' or 'metadata'), a service name (eg 'obs', find, or 'con')
       and a set of parameters as a Python dictionary, return a Python dictionary containing the result.
    """
    if params:
        data = urllib.urlencode(params)  # Turn the dictionary into a string with encoded 'name=value' pairs
    else:
        data = ''
    # Get the data
    try:
        result = json.load(urllib2.urlopen(BASEURL + servicetype + '/' + service + '?' + data))
    except urllib2.HTTPError as error:
        print "HTTP error from server: code=%d, response:\n %s" % (error.code, error.read())
        return
    except urllib2.URLError as error:
        print "URL or network error: %s" % error.reason
        return
    # Return the result dictionary
    return result

#### Python 3.X code, courtesy of Nick Swainston ####################
import urllib.request
```

```
import json

# Append the service name to this base URL, e.g. 'con', 'obs', etc.
BASEURL = 'http://ws.mwatelescope.org/'

def getmeta(servicetype='metadata', service='obs', params=None):
    """Given a JSON web servicetype ('observation' or 'metadata'), a service name (eg 'obs', find, or 'con')
       and a set of parameters as a Python dictionary, return a Python dictionary containing the result.
    """
    if params:
        # Turn the dictionary into a string with encoded 'name=value' pairs
        data = urllib.parse.urlencode(params)
    else:
        data = ''

    # Get the data
    try:
        result = json.load(urllib.request.urlopen(BASEURL + servicetype + '/' + service + '?' + data))
    except urllib.error.HTTPError as err:
        print("HTTP error from server: code=%d, response:\n %s" % (err.code, err.read()))
        return
    except urllib.error.URLError as err:
        print("URL or network error: %s" % err.reason)
        return

    # Return the result dictionary
    return result
##################################################################

# Get and print observation info:

# Some example query data:
fname = '1095675784_20140925102450_gpubox23_01.fits'
starttime = 1095686136

obsinfo1 = getmeta(service='obs', params={'obs_id':starttime})
obsinfo2 = getmeta(service='obs', params={'filename':fname})

coninfo1 = getmeta(service='con', params={'obs_id':starttime})
coninfo2 = getmeta(service='con', params={'filename':fname})

olist = getmeta(service='find', params={'mintime':1097836168, 'maxtime':1097840480, 'obsname':'3C444%'})

faultdict = getmeta(servicetype='observation', service='errors', params={'obs_id':1199602952, 'dictformat':1})
```

The `urlopen()` call above returns a file object (supporting .read(), .readlines(), etc). The above function calls json.load() on that file object to read the contents and convert it to a python dictionary. If you are using the 'fits' or 'skymap' services, or the 'errors' service without specifying 'raw', 'cooked' or 'dictformat' output, the urllib2.urlopen() call returns a file object containing the contents of the FITS, PNG or HTML file, so you do **not** use json.load() on it. Instead, adapt the above function to simply call read() on the file object and return the string. You can then save it to disk, or in the case of the FITS service, pass it to the pyfits.HDUList.fromstring() method to create a FITS structure in memory.

The above code is in the `MandC_Core` git repository, as `mwatools/metaexample.py`