

# The PDS Universal Planetary Coordinates (UPC) Database Export

## What is the Universal Planetary Coordinates (UPC)?

The Universal Planetary Coordinates (or UPC)[1] is a database of many of the level 1 imaging data products archived in the PDS Imaging Node[2]. The UPC includes the camera statistics, URL's for thumbnail and browse images, and the GIS footprint for each image. These data products and meta data are calculated using ISIS3[3,4]. For this reason, only data products which have an ISIS3 camera model can be included in the UPC.

## Which database download do I need?

There are multiple download choices for the UPC database to limit the size of the downloads. They are separated out by the planetary targets which are included in the database. Here are the database downloads and the data they contain:

[Should these be links to the downloads?]

- `upc_search_mars` - This database includes the Martian system (Mars, Phobos, and Deimos)
- `upc_search_moon` - This database includes data for the Moon and Earth
- `upc_search_other` - This database includes data for all planetary targets not included in the above database downloads

## What files are included in the download?

- `help.pdf` - this document
- `upc_search_mars_export.sql` - the PostgreSQL dump of the UPC database, the name will match the download you've chosen
- `cluster_up.sql` - a script to rebuild indexes and cluster the database tables for fast searches
- `allstats.sql` - a script which calls other SQL scripts. These scripts create statistical summaries of the data which can be useful in reviewing the overall data.

## What do I need to use the database download?

You will need a PostgreSQL database server (<http://www.postgresql.org/>) at least version 9.1.4 with the most recent PostGIS (<http://postgis.org/>) extensions for your version of the PostgreSQL database.

## How do I load the database?

### Create a database instance with the PostGIS extensions

You will need a PostgreSQL database instance with the PostGIS extensions included. If a template for PostGIS has been created you can use this to create a new instance with the PostGIS data types and functions included. Otherwise, you will need to apply these changes to your database from the PostGIS repository on your server.

### Loading the data from the download

Once you have a database instance with the PostGIS extensions, you can import the dump file using the psql command:

```
> psql -U [your DB owner] [your DB instance name] < upc_search_mars_export.sql
```

## Why and How should I tune the database after loading?

### Why tune the database?

After the initial load of the database, the data records will be on the hard drive in no particular order. The tuning scripts provided will rebuild key indexes and then use those indexes to sort the data records and rewrite them to the hard drive. This greatly increases the performance as each read of the database is more likely to have the records matching your search.

### How do I tune the database?

To tune the database, run the cluster\_up.sql script after the data has been loaded. This script will vacuum the database, reindex key tables, and then cluster those tables on the indexes. To run the script:

```
> psql -U [your DB owner] [your DB instance name] < cluster_up.sql
```

# How do I search the database with SQL queries?

## Database Schema

Here is a brief description of the tables included in the database and the columns they contain:

- **targets\_meta** - This table defines the planetary bodies that could be imaged. The primary key for this table is `targetid` which is used to link the target to the datafiles entries for a given PDS image. Note, planetary bodies included in this table may not have any associated data depending on which download you chose (see above for explanation of the downloads).
  - `targetid` - integer (primary key)
  - `naifid` - integer (the NAIF ID for the target)
  - `targetname` - varchar (the name of the target in all caps)
  - `system` - varchar (the target system in all caps)
  - `displayname` - varchar (the name of the target in mixed case)
  - `aaxisradius` - double precision (the a axis radius of the target in km)
  - `baxisradius` - double precision (the b axis radius of the target in km)
  - `caxisradius` - double precision (the c axis radius of the target in km)
  - `description` - varchar (currently this field is not used)
  - `iau_mean_radius` - double precision (for targets the IAU has defined a mean radius, the value to be used in place of the A,B, and C radius for projections)
- **instruments\_meta** - This table defines the different instruments images could have been produced by. The primary key for this table is `instrumentid` which is also used to link to the datafiles table.
  - `instrumentid` - integer (primary key)
  - `instrument` - varchar (the instrument name from image labels)
  - `displayname` - varchar (the instrument name in a more readable form for display)
  - `mission` - varchar (the name of the mission in a more readable form)
  - `spacecraft` - varchar (the name of the instrument spacecraft from image labels)
  - `description` - varchar (any notes about the instrument)
  - `product_type` - varchar (whether the image is an EDR or RDR data product)

*Note - there is an instrument defined (instrument ID = 1) which defines common keywords for all instruments.*
- **datafiles** - The datafiles table is the primary table for PDS image data. The primary key for this table is UPC ID. Also included in this table is the download URL for the image and other key search terms.
  - `upcid` - bigint (primary key)
  - `isid` - varchar (the ID created by ISIS)
  - `productid` - varchar (the product ID from the image labels)
  - `edr_source` - varchar (the URL to download the raw image from the PDS)

- `edr_detached_label` - varchar (the URL to download the detached label file, if one exists)
  - `instrumentid` - integer (foreign key references `instruments_meta` to specify which instrument recorded the image)
  - `targetid` - integer (foreign key reference `targets_meta` to specify which target the image is on)
- `keywords` - The `keywords` table defines all the common and instrument specific keywords and their data types. The primary key is the `typeid`. The values for these keywords are captured in the following `meta_*` tables.
  - `typeid` - integer (primary key)
  - `instrumentid` - integer (foreign key references `instruments_meta`)
  - `datatype` - varchar (specifies if the keyword is 'string', 'integer', 'boolean', 'double', 'time', 'geometry')
  - `typename` - varchar (the name of the instrument keyword from the image labels)
  - `displayname` - varchar (a human readable version of the keyword name)
  - `description` - varchar (any notes or further details about the keyword)
  - `shapecol` - varchar (the name used for the keyword in Shapefile exports)
  - `unitid` - integer (foreign key to the units table)
- `meta_geometry` - The `meta_geometry` table stores the center point and footprint GIS geometries for the PDS images.
  - `upcid` - bigint (foreign key references `datafiles` table)
  - `typeid` - integer (foreign key references the `keywords` table)
  - `value` - geometry (the GIS geometry, currently this will either be the center point of the image or the full footprint polygon, both are stored for each image)
- `meta_integer` - The `meta_integer` table stores the values for integer camera keywords such as orbit number or total pixels.
  - `upcid` - bigint (foreign key references `datafiles` table)
  - `typeid` - integer (foreign key references the `keywords` table)
  - `value` - integer (the keyword value)
- `meta_boolean` - The `meta_boolean` table stores true and false values for camera keywords such as whether the image contains the North Pole.
  - `upcid` - bigint (foreign key references `datafiles` table)
  - `typeid` - integer (foreign key references the `keywords` table)
  - `value` - boolean (the keyword value)
- `meta_precision` - The `meta_precision` table store double precision keyword values such as DN mean value or the center emission angle.
  - `upcid` - bigint (foreign key references `datafiles` table)
  - `typeid` - integer (foreign key references the `keywords` table)
  - `value` - double precision (the value of the keyword)
- `meta_string` - The `meta_string` table stores string values for camera keywords such as the mission name.
  - `upcid` - bigint (foreign key references `datafiles` table)
  - `typeid` - integer (foreign key references the `keywords` table)

- value - varchar (the value of the keyword)
- meta\_time - The meta\_time table stores dates and time camera keywords such as the stop and start times.
  - upcid - bigint (foreign key references datafiles table)
  - typeid - integer (foreign key references the keywords table)
  - value - timestamp w/o time zone (the value of the keyword)
- meta\_bands - The band filters and center wave frequencies for the image.
  - upcid - bigint (foreign key references datafiles table)
  - filter - varchar (the name of the filter for the band)
  - centerwave - double precision (the center wave frequency for this band)

## Joining tables to create queries

The way the tables are laid out, you will build your SQL queries around the datafiles table and join the other tables or subqueries to get the results you are looking for. Here's an example query that returns the download URL, the image footprint geometry and the solar longitude:

```
> SELECT d.edr_source, g.value AS geom, l.value AS solarlon
> FROM datafiles d
> JOIN (SELECT i.upcid, i.value FROM meta_geometry i
>       JOIN keywords gk ON (i.typeid=gk.typeid) WHERE
>       gk.typeid='isisfootprint') AS g ON (g.upcid=d.upcid)
> JOIN (select s.upcid, s.value FROM meta_precision s
>       join keywords lk on (s.typeid=lk.typeid) WHERE
>       lk.typeid='solarlongitude') as l ON (l.upcid=d.upcid);
```

In the above example query, we have 2 subqueries which select the meta values we are interested in, the isisfootprint from the meta\_geometry table and the solarlongitude from the meta\_precision table. These subqueries select the value and the UPC ID, which is used to match these results with the datafiles table and the other subqueries. These subqueries join the meta value table with the keywords table so that the keyword can be selected by name. Alternatively, you could look up the typeid value ahead of time and leave out the join with the keywords table. Note, when you are looking for results within a value range, adding the 'where' clause for this check to the subquery will help reduce the result sets from the subquery and can improve query performance.

## Using the PostGIS functions to search by locations

Building on the query above, here is an example that uses a bounding box to limit the search results:

```
> SELECT d.edr_source, g.value AS geom, l.value AS solarlon
> FROM datafiles d
```

```

> JOIN (SELECT i.upcid, i.value FROM meta_geometry i
>       JOIN keywords gk ON (i.typeid=gk.typeid) WHERE
> gk.typeid='isisfootprint' AND
> ST_Intersects(i.value, ST_GeomFromText('POLYGON((10 0, 20 0, 20 10,
10 10, 10 0))', -1))) > AS g ON (g.upcid=d.upcid)
> JOIN (select s.upcid, s.value FROM meta_precision s
>       join keywords lk on (s.typeid=lk.typeid) WHERE
> lk.typeid='solarlongitude') as l ON (l.upcid=d.upcid);

```

This update to the previous query will restrict search results to the images which have a footprint which intersects the bounding box defined by the ST\_GeomFromText function.

## Accessing the data from ArcMap or other GIS tools

Some tools, such as ArcMap, know how to access a PostGIS database, however these tools generally are limited in their ability to create SQL queries. For this reason, it is useful to create either a view or a single table in the database which includes the data you want to work with in ArcMap. The wms\_view.sql script creates a table which is compatible with ArcMap and can be exported as a Shapefile using the GDAL[5] tool ogr2ogr. One thing to note in the wms\_view.sql is the inclusion of the "oid" column. This is an internal column which is not normally returned on queries, but which is required by tools such as ogr2ogr to ensure uniqueness in the table results.

## References

- [1] Akins, S.W., et. al. (2009) LPSC XL, abstract #2002  
(<http://www.lpi.usra.edu/meetings/lpsc2009/pdf/2002.pdf>)
- [2] Planetary Data System Standards Reference, v. 3.8, JPL D-7669, Part 2  
(<http://pds.nasa.gov/tools/standards-reference.shtml>)
- [3] Keszthelyi, L., et al., 2013, LPSC XLIV, abstract #2546  
(<http://www.lpi.usra.edu/meetings/lpsc2013/pdf/2546.pdf>)
- [4] Sides S. et al., 2013, LPSC XLIV, abstract #1746  
(<http://www.lpi.usra.edu/meetings/lpsc2013/pdf/1746.pdf>)
- [5] Geospatial Data Abstraction Library (<http://www.gdal.org/>)