International
Virtual
Observatory
Alliance

HATS: A Standard for the Hierarchical Adaptive Tiling Scheme in the Virtual Observatory

Version 1.0

IVOA Note ???

Working Group
Applications Group

This version
https://www.ivoa.net/documents/Notes/hats-ivoa/???

Latest version
https://www.ivoa.net/documents/Notes/hats-ivoa

Previous versions
This is the first public release

Author(s)
Neven Caplar, Melissa DeLucchi, Sean McGuire, Sandro Campos, Derek Jones, Doug Branton, LINCC team...

Editor(s)
editor here

Version Control
Revision 18cb591-dirty, 2025-04-21 11:07:26 -0400

Abstract

The increasing complexity and volume of astronomical datasets necessitate efficient spatial indexing and query strategies within the Virtual Observatory (VO). The Hierarchical Adaptive Tiling Scheme (HATS) is a framework designed to facilitate scalable queries, filtering operations, and efficient data retrieval across large astronomical surveys. Traditional spatial indexing methods often struggle with the massive scale of modern astronomical datasets, leading to inefficient query execution and storage overhead. HATS provides a flexible, hierarchical approach that balances computational efficiency and adaptability to non-uniform data distributions.

This document describes the structure, implementation, and best practices for integrating HATS within the VO ecosystem, ensuring interoperability and performance optimization for distributed astronomical datasets. The reference implementation of HATS can be found at https://github.com/astronomy-commons/hats. Additionally, we discuss how HATS enhances existing indexing schemes, its role in federated data access, and its potential applications for time-domain astronomy, large-scale surveys, and cross-matching of astronomical catalogs.

TODO: we talk about time-domain astronomy here, but there's no elaboration later.

Status of this document

This is an IVOA Note expressing suggestions from and opinions of the authors. It is intended to share best practices, possible approaches, or other perspectives on interoperability with the Virtual Observatory. It should not be referenced or otherwise interpreted as a standard specification.

A list of current IVOA Recommendations and other technical documents can be found in the IVOA document repository[1].

Contents

[1] https://www.ivoa.net/documents/

Acknowledgments

The authors thank the IVOA Applications Working Group and various contributors from the astronomical community for their feedback and discussions that shaped this standard. We acknowledge the key collaborations with Space Telescope Science Institute (STScI) and IPAC, whose expertise and contributions have been invaluable in refining the HATS framework. Additionally, we extend our gratitude to the Strasbourg astronomical Data Center (CDS) for their assistance in metadata structuring and interoperability support. This work has also benefited from insights provided by the S-Plus survey, Linea, European Space Agency, and Canadian Astronomy Data Center (CADC), whose contributions have helped enhance the applicability and robustness of HATS in large-scale astronomical data analysis. TODO: Tidy the author list before sending for internal review.

Conformance-related definitions

The words “MUST”, “SHALL”, “SHOULD”, “MAY”, “RECOMMENDED”, and “OPTIONAL” (in upper or lower case) used in this document are to be interpreted as described in IETF standard RFC2119 (Bradner, 1997).

The *Virtual Observatory (VO)* is a general term for a collection of federated resources that can be used to conduct astronomical research, education, and outreach. The International Virtual Observatory Alliance (IVOA) is a global collaboration of separately funded projects to develop standards and infrastructure that enable VO applications.

1 Introduction

The rapid expansion of astronomical data from large survey facilities like Vera C. Rubin Observatory, Euclid, and Roman Space Telescope necessitates innovative solutions for spatial indexing and efficient data retrieval. These surveys generate vast amounts of high-resolution imaging and time-domain data, requiring efficient methods for organizing, querying, and cross-matching data across multiple archives. Traditional approaches to spatial indexing, such as hierarchical pixelization (e.g., HEALPix) or static tiling schemes, often exhibit inefficiencies when handling dynamic, multi-resolution datasets.

The Hierarchical Adaptive Tiling Scheme (HATS) is a novel approach designed to optimize spatial data partitioning while maintaining flexibility in accommodating varying data densities. Unlike fixed spatial partitioning methods, HATS dynamically adjusts tile sizes based on local data characteristics, ensuring an optimal balance between resolution, query efficiency, and storage management. By leveraging a hierarchical structure, HATS allows users to perform efficient multi-resolution queries while preserving high precision in regions of interest.

This document aims to define best practices for implementing and utilizing this tiling scheme within the Virtual Observatory framework. This document outlines the principles behind HATS, describes its data model, and provides recommendations. Additionally, we discuss how HATS can facilitate efficient cross-matching of astronomical catalogs, accelerate large-scale spatial queries, and enhance interoperability between diverse astronomical datasets.

2 Motivation and Goals

The primary motivation behind HATS is to address the following challenges in astronomical data management:

- **Scalability:** Modern astronomical surveys generate petabytes of spatially distributed data, requiring an indexing scheme that scales efficiently with dataset size. HATS provides files that are similarly-sized, and are well-suited to parallel operations. NEVEN - review! each of the later three bullets says how HATS addresses the problems. similar style added here.

- **Adaptive Resolution:** Fixed grid-based partitioning often leads to inefficient storage and query execution, particularly in non-uniformly distributed datasets. HATS dynamically adjusts tile sizes to accommodate varying data densities.

- **Efficient Query Execution:** Spatial queries such as nearest-neighbor searches and cross-matching must be executed efficiently across distributed data repositories. HATS enables rapid indexing and retrieval of relevant data subsets.

- **Interoperability:** Astronomical data is collected from diverse instruments and observatories, often using different spatial reference frames. HATS provides a standardized framework for integrating and harmonizing spatial data across multiple sources.

3 HATS Design and Implementation

3.1 HATS Catalog Directory Structure

The HATS framework relies on spatially sharding catalogs of approximately the same size in Parquet files. Here, we discuss how this is achieved and additional concepts that make it easier to use this main idea for astronomical research.

The central unit of data storage is the HATS `catalog`. It stores the data along with the associated metadata needed to access it. The `catalog` organization structure is shown in Listing 1.

```
catalog/
|-- [REQUIRED] properties
|-- [RECOMMENDED] partition_info.csv
|-- [OPTIONAL] point_map.fits
|-- [OPTIONAL] data_thumbnail.parquet
+-- dataset/
    |-- [RECOMMENDED] _metadata
    |-- [RECOMMENDED] _common_metadata
    |-- Norder=0/
    |-- Norder=1/
    |-- Norder=2/
    |-- Norder=3/
    |-- Norder=4/
    |-- Norder=5/
    |-- Norder=6/
    +-- Norder=. . ./
```

Listing 1: Example catalog directory contents

The astronomy data is stored in the directory `dataset`, and further, within the subdirectories that specify the order at which particular part of the dataset is stored. We will discuss the partitioning and data storage in Sections 3.1.1, 3.1.2 and 3.1.3. The other files visible above are various metadata and auxiliary files that are here to enable better and easier handling of the data and we will describe them in Section 3.3.

3.1.1 Hierarchical Directory Structure

Focusing now on the dataset's contents, HATS employs a multi-level hierarchy based on HEALPix tiling, where each level represents a progressively finer spatial resolution.

All tiles of the same HEALPix order are contained within the same prefix `Norder=k` directory. To avoid directories becoming too large for some file systems, the tiles are then grouped by a `Dir` subdirectory prefix, where the value of the `Dir` key is the result of integer division by 10,000 of the pixel number.

We see the directory structure in Listing 2, showing a dataset with leaf parquet files at several HEALPix orders.

```
dataset/
|-- . . .
|-- Norder=6/
|   |-- Dir=0/
|   |   |-- Npix=0.parquet
|   |   |-- . . .
|   |   +-- Npix=9999.parquet
|   +-- Dir=10000/
|       +-- Npix=10000.parquet
|           +-- . . .
|-- Norder=7/
+-- . . .
```

Listing 2: Example catalog dataset directory contents

The data is stored in the parquet files (discussed in Section 3.1.3), with one or multiple files being possible in the final directory, i.e., in the ultimate data leaf.

If there are multiple files, they should be read together, i.e., we consider them to be one single data unit. In this way, small updates can be added to already existing `catalogs` with simple, correctly placed, additions of files in existing folders.

Such a directory structure would appear as shown in Listing 3.

```
dataset/
|-- . . .
|-- Norder=6/
|   |-- Dir=0/
|   |   |-- Npix=0/
|   |   |   |-- part0.parquet
|   |   |   +-- . . .
|   |   +-- Npix=9999/
|   |       |-- part0.parquet
|   |       +-- . . .
|   +-- Dir=10000/
|       +-- Npix=10000/
|           |-- part0.parquet
|           +-- . . .
|-- Norder=7/
+-- . . .
```

Listing 3: Example catalog dataset directory contents with leaf directories

3.1.2 Adaptive Tiling Algorithm

Unlike static partitioning schemes, HATS dynamically subdivides spatial regions based on data density. In areas with sparse data, larger tiles minimize storage overhead, whereas high-density areas are subdivided into smaller tiles to improve query efficiency.

The data is stored at a given level until the dataset size crosses a predetermined threshold. This threshold can be, most commonly, the number of rows or the size of the data on the disk. At this point, the data gets split into four higher-order HEALPix tiles using the spatial information contained in the data. This process continues until all of the data is stored at the appropriate level and no data leaf has more data than the predetermined threshold.

TODO: Melissa thinks description this would benefit from a figure.

3.1.3 Structure of Data Files

The astronomical data is stored in `*.parquet` format. TODO - add a footnote or citation for parquet Parquet is a columnar storage file format optimized for efficient data compression and retrieval, especially well-suited for analytical workloads. It is ideal for storing large amounts of astronomical

tabular data because it allows fast access to specific columns without reading the entire dataset, significantly reducing I/O and improving performance.

The HATS format RECOMMENDS that the first column of the dataset is the `healpix_29` index column. `healpix_29` stores the crucial spatial information about the position in the sky for each row, and it is not unique. This is calculated as the HEALPix order 29 value of the row's right ascension and decliation. If two objects occur at the same location (or the data is individual observations of the same sky object), then multiple rows may have the same value for the `healpix_29` column. The existence of this value speeds up downstream spatial calculations, and is beneficial for spatially-intensive applications.

Additional optional performance considerations for Parquet files is discussed in Section 4.1.

3.2 Supplemental Tables

To improve scalability and efficient query optimization, HATS readers can benefit from alternative arrangements of the data. We refer to these as supplemental tables or supplemental catalogs, and these offer a limited or re-projected view of the primary dataset.

3.2.1 Margin Cache

One of primary motivations for HATS is rapid cross-matching of different surveys, or different data sources within the same survey. The power of cross-matching in HATS is that each computation unit gets smaller spatially-connected parts from each dataset and works on them one part at the time.

However, this introduces a limitation: at the boundaries of a divided section, some data points that should be cross-matched will be missed because they are present just across the border in neighboring partitions instead. This has a number of causes (observational error, non-point source positions, proper motion, etc), but should be handled by applications to ensure that you're finding an appropriate counterpart in cross-match.

One option would be to load all neighboring partitions when performing a cross-match. However, this would require loading much unnecessary data, as only a small sliver around the edge of a partition is a candidate for counterparts in neighboring partitions. Figure 1 shows some example data points that are near to a HEALPix pixel boundary, and should be included in a cross-match calculation.

We address this through the creation of a margin cache. This is an additional catalog that contains points in a limited angular threshold around the primary catalog partition. The margin data can be loaded alongside the primary catalog partition data during a cross-match operation to account for

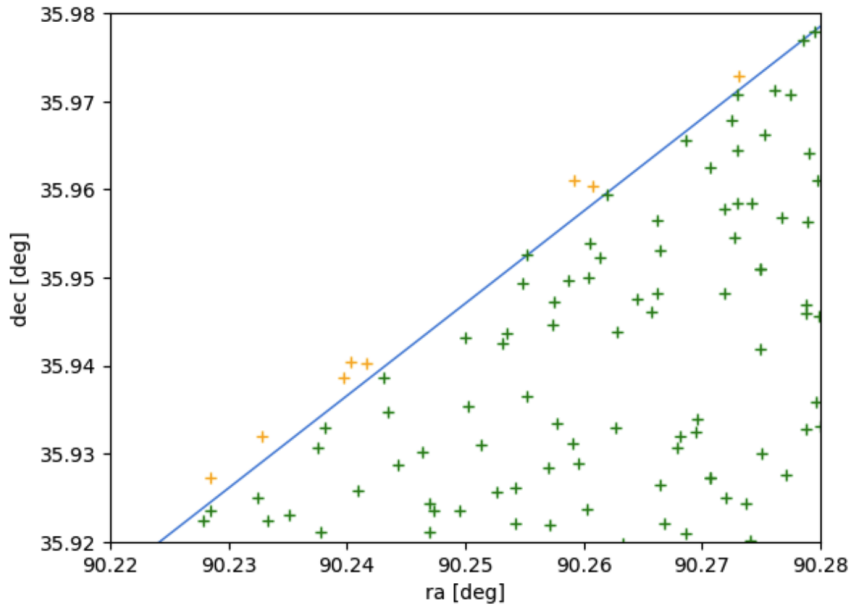

Figure 1: Example of margin contents. Green points are present in the primary catalog partition, the blue line is the HEALPix pixel boundary, and yellow points are points in the neighboring pixel that are within 10 arcseconds of the boundary.

any positional issues that may arise around the boundaries of the partition's HEALPix pixel.

3.2.2 Index Table

HATS catalogs are partitioned spatially, on right ascension and declination. This makes finding objects in a particular area of the sky very straightforward. However, you may occasionally only know the object by the survey-assigned identifier, and to find the row, you would have to perform a full scan of all partitions.

HATS supports creating additional secondary index tables. In their creation, you perform this full scan and shuffle and write out a simple mapping from the unique identifier to where it can be found in the sky. In this way, you can query the index catalog for the HEALPix pixel where the value can be found, and then need only load the primary catalog partition for the corresponding HEALPix pixel. This greatly speeds up and facilitates the query, as we now have to load only single pixel instead of going through the whole catalog. These are general secondary indexes, and can be built on any column, not just the survey-assigned identifier.

3.2.3 Catalog Collection

Margins and indexes are associated with a single astronomical dataset, and to make this connection clearer and enable friendlier application behavior, tables MAY be grouped together under a single directory called a catalog collection. These include the primary data `catalog` and other `catalogs` that are optional and are intended either to improve access to the main `catalog` or to enrich it with additional information.

In Listing 4, we present an overview of this folder structure, including a few common examples of such optional catalogs. The catalog collection directory MUST contain a `collection.properties` file, whose content is outlined in Section 3.3.7.

```
gaia_dr3/
|-- collection.properties
|-- catalog/
|   |-- properties
|   +-- . . .
|-- [OPTIONAL] margin_5arcs/
|   |-- properties
|   +-- . . .
|-- [OPTIONAL] margin_10arcs/
|   |-- properties
|   +-- . . .
+-- [OPTIONAL] index_designation/
    |-- properties
    +-- . . .
```

Listing 4: Example collection directory contents

TODO: we talk about association catalog in Table 2, but we dont mention it anywhere

3.2.4 Association Table

Noting again that cross-matching and multi-survey analysis is a primary motivator for the HATS spatial format, we introduce the HATS association table.

This table provides a spatially-sharded set of links between rows in two different object catalogs. This mirrors the relational database notion of a link table, and will likely be composed of the primary survey identifiers from either side of the association. We make no prescription as to whether the relationships are one-to-one or one-to-many.

There is a single "primary" catalog that reflects the left side, and the other catalog is the "join" catalog (as the data can be joined into the primary catalog). The resulting association data will use the coordinates of the primary catalog, and MAY be partitioned with the same pixels as the primary catalog. If there are many more or many fewer association links than there are objects in the primary table, it may be preferable to either combine or split the leaf parquet files further.

3.2.5 Map Catalog

TODO: we should mention this, as it's very useful for science.

3.3 Metadata and Auxiliary Files

HATS implementations utilize auxiliary files and metadata files to store relevant information about the structure, including:

- **[REQUIRED] properties**, at `catalog` level
- **[RECOMMENDED] partition_info.csv**, at `catalog` level
- **[RECOMMENDED] partition_join_info.csv**, at `catalog` level
- **[OPTIONAL] point_map.fits**, at `catalog` level
- **[OPTIONAL] data_thumbnail.parquet**, at `catalog` level
- **[RECOMMENDED] _metadata**, at `catalog/dataset` level
- **[RECOMMENDED] _common_metadata**, at `catalog/dataset` level
- **[REQUIRED] collection.properties**, at `catalog` collection level

Note that collection.properties is only required if you are creating `catalog` collection for your dataset.

We will now go over these files and explain their function, format and their contents.

3.3.1 properties

A text file named `properties` is REQUIRED in the root level of the catalog directory. It marks the directory as containing a HATS catalog, and so MUST be located in the root directory of the catalog. It MUST be encoded in UTF-8, with one line per property, following the syntax `keyword = value`. The ordering of the keywords is not important. The keywords MAY include many of those listed in Table 1.

The text file may contain comment lines, beginning with the `'#'` character.

An example `properties` file is shown in Listing 5.

```
#HATS catalog
obs_collection=euclid_q1_merFinalCatalog
dataproduct_type=object
hats_nrows=29767806
hats_col_ra=RIGHT_ASCENSION
hats_col_dec=DECLINATION
hats_cols_sort=OBJECT_ID
hats_max_rows=1000000
hats_order=6
moc_sky_fraction=0.00618
hats_builder=hats-import v0.4.4
hats_creation_date=2025-03-20T03\:38UTC
hats_estsize=23137775
hats_release_date=2024-09-18
hats_version=v0.1
```

Listing 5: Example `properties` file contents

We enforce additional requirements for the presence of particular fields for different types of HATS tables. A matrix of these requirements is shown in Table 2.

HATS Keyword	**Description - Format - Example**
addendum_did	If content has been added after initial catalog creation, creator_did of any added data
all_indexes	For catalog collections, space-delimited map of indexed field to subdirectories containing index tables.
all_margins	For catalog collections, space-delimited list of subdirectories containing margin caches.
bib_reference	Bibliographic reference
bib_reference_url	URL to bibliographic reference
creator_did	Unique ID of the HATS - Format: IVOID - Ex : ivo://CDS/P/2MASS/J
data_ucd	UCD describing data contents
dataproduct_type	Format: one word ONE OF(object, margin, association, index, ...)
default_index	For catalog collections, the field of the default index to use for ID searches.
default_margin	For catalog collections, the subdirectory containing the default margin cache to use for cross-matching
hats_assn_join_table_url	For association tables, there will be a join table with original survey data (right side of the join)
hats_assn_leaf_files	For association tables, does the table contain leaf files (may optionally only provide a "soft" association between tiles only).
hats_builder	Name and version of the tool used for building the HATS. Format: free text – Example "hats-import v0.6.4"
hats_col_assn_join	For association tables,column name for the joining (right) side of the join within the original table
hats_col_assn_join_assn	For association tables, column name in the assocation table for the join (right) side of the association table.
hats_col_assn_primary	For association tables, column name for the primary (left) side of the join
hats_col_assn_primary_assn	For association tables, column name in the association table that matches the primary (left) side of the join

HATS Keyword	Description - Format - Example
hats_col_dec	Column name of the dec coordinate. Used for partitioning and default cross-matching.
hats_col_ra	Column name of the ra coordinate. Used for partitioning and default cross-matching.
hats_cols_default	Which columns should be read from parquet files, when user doesn't otherwise specify. Useful for wide tables. Format: space-delimited column names
hats_cols_sort	At catalog creation time, the columns used to sort the data, in addition to `healpix_29` column.
hats_cols_survey_id	The primary key used in the original survey data. May be multiple columns if the survey uses a composite key (e.g. object ID and MJD for detections)
hats_coordinate_epoch	For the default ra and dec (hats_col_ra, hats_col_dec), the measurement epoch
hats_copyright	Copyright mention associated to the HATS - Format: free text
hats_creation_date	HATS first creation date - Format: ISO 8601 => YYYY-mm-ddTHH:MMZ
hats_creator	Institute or person who built the HATS. Format: free text. Ex : CDS (T.Boch)
hats_estsize	HATS size estimation. Format: positive integer. Unit : KB
hats_frame	Coordinate frame reference. Format: word "equatorial" (ICRS), "galactic", "ecliptic"
hats_index_column	For index tables, the column that is indexed over
hats_index_extra_column	For index tables, extra columns that are carried through with the index
hats_margin_threshold	For margin tables, the threshold used for finding points within margin. Units: arcs
hats_max_rows	At catalog creation time, the maximum number of rows per file before breaking into 4 new files at higher order.
hats_nrows	Number of rows of the HATS catalog. Format: positive integer
hats_order	Deepest HATS order. Format: positive integer
hats_primary_table_url	For supplemental tables , there will be a primary table with original survey data.
hats_progenitor_url	URL to an associated progenitor HATS
hats_release_date	Last HATS update date - Format: ISO 8601 => YYYY-mm-ddTHH:MMZ
hats_service_url	HATS access url. Format: URL
hats_status	HATS status. Format: list of space-delimited words ("private" or "public"), ("main", "mirror", or "partial"), ("clonable", "unclonable" or "clonableOnce"). Default : public main clonableOnce
hats_version	Number of HATS version. Format: 0.1 (corresponds to version of this note)
moc_sky_fraction	Fraction of the sky covered by the MOC associated to the HATS. Format: real between 0 and 1
npix_suffix	String to indicate file suffix for leaf files. In the typical HATS directory structure, this is `'.parquet'` or `'.pq'` because there is a single file in each Npix partition. If using leaf directories, `'/'`.
obs_ack	Acknowledgment mention.

HATS Keyword	Description - Format - Example
obs_collection	Short name of original data set. Format: one word. Ex : 2MASS
obs_copyright	Copyright mention associated to the original data. Format: free text
obs_copyright_url	URL to a copyright mention
obs_description	Data set description. Format: free text, longer free text description of the dataset
obs_regime	General wavelength. Format: word: "Radio" | "Millimeter" | "Infrared" | "Optical" | "UV" | "EUV" | "X-ray" | "Gamma-ray"
obs_title	Data set title. Format: free text, one line. Ex : HST F110W observations
prov_progenitor	Provenance of the original data. Format: free text
publisher_id	Unique ID of the HATS publisher. Format: IVOID - Ex : ivo://CDS
t_max	Stop time of the observations. Format: real. Representation: MJD
t_min	Start time of the observations. Format: real. Representation: MJD

Table 1: Available keys for properties file

HATS Keyword	HATS Catalog Type				
	object	margin	index	association	collection
all_indexes					opt
all_margins					opt
dataproduct_type	REQ	REQ	REQ	REQ	
default_index					opt
default_margin					opt
hats_assn_join_table_url				REQ	
hats_assn_leaf_files				REQ	
hats_col_assn_join				REQ	
hats_col_assn_join_assn				opt	
hats_col_assn_primary				REQ	
hats_col_assn_primary_assn				opt	
hats_col_dec	REQ	opt			
hats_col_ra	REQ	opt			
hats_cols_default	opt	opt			
hats_index_column			REQ		
hats_index_extra_column			opt		
hats_margin_threshold		REQ			
hats_npix_suffix	opt	opt	opt	opt	
hats_nrows	REQ	REQ	REQ	REQ	
hats_primary_table_url		REQ	REQ	REQ	REQ
obs_collection	REQ	REQ	REQ	REQ	REQ

Table 2: Catalog-type specific fields. For display, REQ is REQUIRED, and opt is OPTIONAL

3.3.2 partition_info.csv

A text file named `partition_info.csv` is OPTIONAL and RECOMMENDED in the root level of the catalog directory. If present, it MUST be a CSV (comma-separated-values) file, with the columns `"Norder"` and `"Npix"`, as shown in example contents in Listing 6. Additional columns might be present in the file, but are not required, and may not be interpreted by all HATS readers. The values of pairs of `"Norder"` and `"Npix"` reflect the HEALPix tiles of the catalog's partitions. HATS readers can quickly read this file to understand the full scope of the catalog, and potential spatial overlap with other catalogs.

```
Norder, Npix
3,      530
4,      637
4,      958
4,      1003
4,      2147
```

Listing 6: Example `partition_info.csv` file contents

3.3.3 partition_join_info.csv

A text file named `partition_join_info.csv` is OPTIONAL and RECOMMENDED in the root level of the catalog directory for an association catalog. If present, it MUST be a CSV (comma-separated-values) file, with the columns `"Norder"`, `"Npix"`, `"join_Norder"`, and `"join_Npix"`, as shown in example contents in Listing 7. Additional columns might be present in the file, but are not required, and may not be interpreted by all HATS readers. The values of pairs of `"Norder"` and `"Npix"` reflect the HEALPix tiles of the primary catalog's partitions. The corresponding values of pairs of `"join_Norder"` and `"join_Npix"` reflect the HEALPix tiles of the join catalog's partitions that have matches inside the primary catalog's partition. HATS readers can quickly read this file to understand the full spatial overlap with other catalogs, and plan which leaf partitions of the primary and join catalogs will be loaded.

This file is not used for table types that are NOT association tables.

```
Norder, Npix,   join_Norder,    join_Npix
3,      530,    3,              517
3,      530,    4,              2120
3,      530,    4,              2121
3,      530,    4,              2122
4,      637,    4,              637
```

Listing 7: Example `partition_join_info.csv` file contents

3.3.4 point_map.fits

A FITS file containing the MOC (multi-order coverage map) of the catalog. This is a two-dimensional histogram of points in each HEALPix tile at some reasonably high order. This will be at the highest order calculated during the catalog ingestion process (likely 9 or 10). This data is useful when inspecting catalogs and understanding the distribution of data.

This file is OPTIONAL and RECOMMENDED for object catalogs.

3.3.5 data_thumbnail.parquet

This is a small dataset aimed to help users to understand and use the data. It is created by taking the first row from each data leaf, so the number of rows in this Parquet file is the same as the number of data leaves altogether.

This file allows a user to get a quick overview of the whole dataset in the same format as the whole dataset. Given how it is sampled, it will cover the entire width of the dataset and give a user an accurate overview of the properties of the dataset. In such a way, it is superior and more convenient than pointing a user to take out a subset of a single Parquet data leaf for testing.

This file is OPTIONAL and RECOMMENDED for object catalogs.

3.3.6 _metadata and _common_metadata

Many parquet reading frameworks support and recommend additional dataset-level metadata files:

- `_common_metadata` which contains the full schema of the dataset, and can be thought of as extensive header information. This file will know all of the columns and their types, as well as any top-level key-value metadata associated with the full parquet dataset.
- `_metadata` contains per-partition information, chiefly the footer information of all constituent parquet files which contain aggregate statistics.

Both files are OPTIONAL and RECOMMENDED for all catalogs.

To understand the importance of these files, it is helpful to understand the structure of partitioned parquet files. Figure 2 shows a schematic of two partitioned parquet files. There is some top-level metadata that may describe the full dataset, as well as column-level key-value metadata. The data values are shown in gray, and typically will take up most of the space of the parquet file. Parquet files will also have footers which may contain aggregate statistics about each column (the min value, max value, count of valid values).

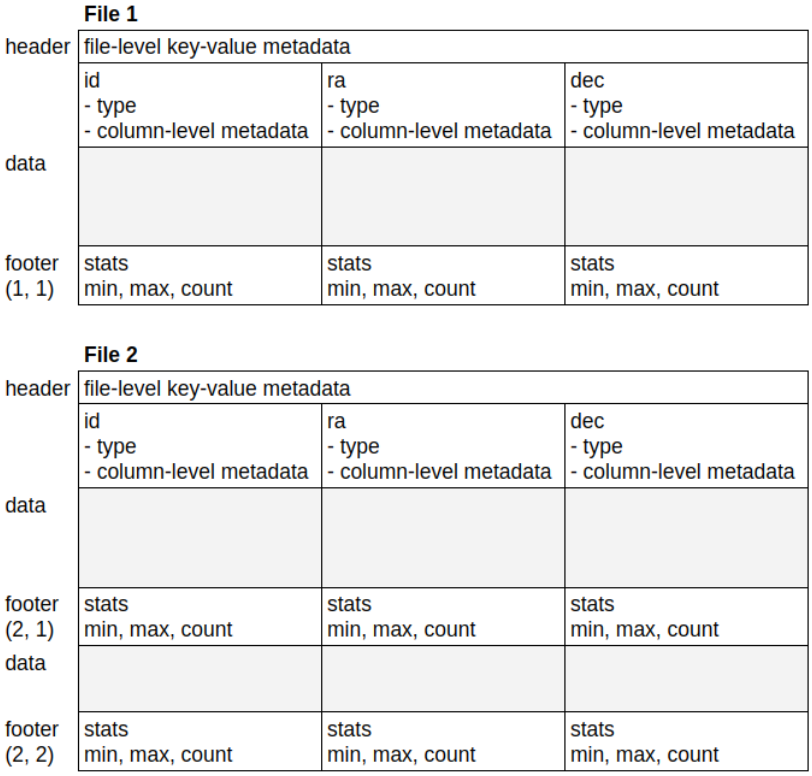

Figure 2: TODO, increase font in the figure Example file layout of two parquet files of a dataset

For very large datasets, it is helpful to have a single file that holds the common header information in all of the partitioned parquet files (assuming that they have homogeneous structure). This `_common_metadata` file can be much smaller (and so faster to read) than a leaf parquet file. The footers, however, will be different for every file, and some files may contain multiple footers if the data inside a single file is very large. These footers can be concatenated into a single `_metadata` file, and can provide valuable insight into the distribution of the data. A clever parquet reader can use this information to filter queries to only those partitions where certain values are possible. See Figure 3 for the layout of these parquet metadata files.

3.3.7 collection.properties

A text file named `collection.properties` is REQUIRED for a catalog collection. It marks the directory as containing a catalog collection, and so MUST be located in the root of the catalog collection. It MUST be encoded in UTF-8, with one line per property, following the syntax `keyword = value`. The ordering of the keywords is not important. The keywords MAY

_common_metadata

header	file-level key-value metadata		
	id - type - column-level metadata	ra - type - column-level metadata	dec - type - column-level metadata

_metadata

header	file-level key-value metadata		
	id - type - column-level metadata	ra - type - column-level metadata	dec - type - column-level metadata
footer (1, 1)	stats min, max, count	stats min, max, count	stats min, max, count
footer (2, 1)	stats min, max, count	stats min, max, count	stats min, max, count
footer (2, 2)	stats min, max, count	stats min, max, count	stats min, max, count

Figure 3: Example file layout of two supplemental parquet files of a dataset

include many of those listed in Table 1, with collection-specific fields shown in Table 2.

The text file may contain comment lines, beginning with the `'#'` character. An example `properties` file is shown in Listing 8.

```
#HATS Collection
obs_collection=gaia_dr3
hats_primary_table_url=gaia
all_margins=gaia_5arcs gaia_1arcs
default_margin=gaia_5arcs
all_indexes=designation designation_index
```

Listing 8: Example `collection.properties` file contents

4 Performance Considerations

Here, we will elaborate on several ways in which this format can be efficiently used. These insights come from our work with LSDB[2], a Python implementation of a package that works natively with HATS catalogs.

4.1 Parquet Storage

Firstly, we emphasize the need to use the Parquet column filtering. This is a standard practice in SQL-like workflows where a user requests only

[2] https://lsdb.io

the columns they need, but it is less common in Python-like workflows. Loading into memory the columns that a user needs for scientific analysis, usually a couple out of tens or hundreds available, significantly reduced the computational requirements for the analysis.

Parquet files can also be split into so-called rowgroups. This is the splitting of Parquet into chunks with a fixed number of rows. Parquet readers can skip over entire row groups if they don't contain relevant data. This can significantly increase the efficiency of particular queries, especially if the rowgroups are selected in a particular way that is appropriate for the scientific case. For instance, if rowgroups are made to be small and sorted by the identification number of the survey, the retrieval of the individual rows by survey identification can be made much faster. This is because we don't have to load the entire Parquet file into memory, only this tiny rowgroup part, in order to retrieve the needed row from the user.

4.2 HTTP Services

The fact that the data can be stored on the hard drive and served at rest to the users simplifies the cost structure for catalog providers. Still, a user operating on the dataset, even if they are doing aggressive filtering and requesting a minimal number of rows at the end, will have to effectively transfer a large amount of data over to their client, where the filtering is done. These limitations could become prohibitive if this is done over a network or with limited bandwidth. To alleviate that problem, it is possible to implement a server-side query in which the filtering operations are done server-side, and only the final dataset is sent to a user. Of course, this requires computational resources on the provider's side, but operate on a single file and will benefit from parquet storage optimizations from Section sec:parquetPerformance

4.3 Cross-matching

Finally, we want to highlight the exceptional performance possible when cross-matching HATS catalogs. Due to its spatial sharding, the cross-matching approach implemented in LSDB is competitive with the existing tools and is more efficient for extensive catalogs, starting with roughly one million rows. Because of the granular spatial structure, the user can increase the number of parallel computation units. These will linearly decrease the time needed as long as the number of workers is smaller than the number of partitions in the datasets and there is sufficient I/O speed. In general, for typical cases of large catalogs (billion+ rows), cross-matching on a single core is around 5 to 15% slower than the pure I/O speed. As discussed above, selecting only specific columns and parallelizing the work can drastically improve performance.

5 Integration with Existing VO Standards

HATS is designed to be compatible with existing VO spatial indexing frameworks, such as HEALPix and MOC (Multi-Order Coverage maps).

TODO, NEED HELP, especially with MOC: explain more how is it compatible.

The HATS format can be made to be compatible with the TAP query by implementing a translation layer between the TAP query language. We have explored some initial implementation of such functionality, but the implementation details will always depend on the language used to handle the Parquet files.

We are closely following the development of the VOParquet format and aim to implement it as a part of HATS catalogs.

A Changes from Previous Versions

No previous versions yet.

References

Bradner, S. (1997), 'Key words for use in RFCs to indicate requirement levels', RFC 2119. http://www.ietf.org/rfc/rfc2119.txt.