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 13c0a84-dirty, 2025-04-11 14:51:25 -0700

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.

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 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, whose contributions have helped enhance the applicability and robustness of HATS in large-scale astronomical data analysis.

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.
- **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

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.

3.1 Catalog collections

The central unit of data storage is HATS `catalog`. It stores the data along with the associated metadata needed to access it. `catalog` will be described in more detail in the following sections (starting with Section 3.2).

At the top level, we organize different possible `catalog` associated with a single astronomical dataset. 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. For instance, one common such `catalog` is the margin `catalog`, containing information about objects near the spatial border of each spatial shard. This dramatically simplifies and improves crossmatching ability when using HATS datasets.

TODO, NEED HELP - Explain the structure of the collection.properties. Explain what the format is, what is contained, what is the obligatory and what is optional information Below, we present an overview of this folder structure, including a few common examples of such optional catalogs:

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

A text file named `collection.properties` must be provided. 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 include many of those listed in the TODO - table of keyword vocabulary.

The additional keywords for the `collection.properties` file to provide linking between catalogs and supporting tables are:

- `name` human-readable name of the collection
- `hats_primary_table_url` subdirectory of the collection for the primary catalog
- `all_margins` space-delimited list of margin cache tables
- `default_margin` the default margin to be used for cross-matching
- `all_indexes` space-delimited list of index tables
- `default_index` the default index to be used for id searches (will typically be the survey identifier)

3.2 Structure of a catalog

TODO, NEED HELP - is the optional designation correct for each one of these metdata files. Melissa - added required/recommended/optional. fixed names Focusing now on an individual `catalog`, the `catalog` organization structure is shown below:

```
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=.../
```

The astronomy data is stored in the directory dataset, within the subdirectories that specify the order at which particular part of the dataset is stored. We will discuss the the partitioning and data storage in Sections 3.3, 3.4 and 3.5. 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 3.6.

3.3 Hierarchical 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 divisiion by 10000 of the pixel number.

We see the following structure, 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/
`-- ...
```

Melissa - changed the example to use actual numbers, and multiple healpix orders. this mirrors page 8 of https://www.ivoa.net/documents/HiPS/20170519/REC-HIPS-1.0-20170519.pdf. The data is stored in the parquet files (discussed in Section 3.5), 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 instead appear like:

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

3.4 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.

3.5 Structure of files

The astronomical data is stored in *.parquet format. 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 index column of the dataset is `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. TODO, NEED HELP please describe how it is calculated; explain how it is not unique when multiple rows at the same HEALPix of order 29 position

3.6 Metadata and Auxiliary Files

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

- **[REQUIRED]collection.properties**, at `catalog` collection level
- **[RECOMMENDED] partition_info.csv:**, at `catalog` level
- **[OPTIONAL] point_map.fits:**, at `catalog` level
- **[OPTIONAL] data_thumbnailparquet**, at `catalog` level
- **[REQUIRED]properties:**, at `catalog` level
- **[RECOMMENDED]_metadata.json:**, at `catalog`/dataset level
- **[RECOMMENDED] _common_metadata:**, at `catalog`/dataset level

3.6.1 hats.properties

TODO, NEED HELP - 1. what does it do 2. what is the format 4. what information does it contain, 4. what is the obligatory part of that information

3.6.2 partition_info.csv

TODO, NEED HELP-1. what does it do 2. what is the format 4. what information does it contain, 4. what is the obligatory part of that information Norder, Npix information here.

3.6.3 point_map.json

TODO, NEED HELP -1. what does it do 2. what is the format 4. what information does it contain, 4. what is the obligatory part of that information

3.6.4 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.

3.6.5 properties

TODO, NEED HELP-1. what does it do 2. what is the format 4. what information does it contain, 4. what is the obligatory part of that information total rows, columns statistics (min, max) , java properties file , ra, dec, default columns, how it was created

3.6.6 metadata.json

TODO, NEED HELP -1. what does it do 2. what is the format 4. what information does it contain, 4. what is the obligatory part of that information It contains per partition information

3.6.7 common_metadata.json

TODO, NEED HELP -1. what does it do 2. what is the format 4. what information does it contain, 4. what is the obligatory part of that informationn Information that is true for everything

3.7 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.

3.8 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, a Python implementation of a package that works natively with HATS catalogs.

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 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.

The fact that the data can be stored on the hard drive and served 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.

Finally, we want to highlight the exceptional performance possible when crossmatching HATS catalogs. Due to its spatial sharding, the crossmatching 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 workers. 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), crossmatching 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. TODO, Add citations

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.