GRASS - GIS

Open Source Geospatial Foundation (OSGeo)

Proposal - Google Summer of Code 2026

Personal Details

- Name: Anirban Das
- IRC and Github nick: Dasux
- Telephone: [redacted]
- Country of Residence: India
- Typical Working Hours:
  - 9:00 - 12:00; 14:00 - 16:00; 18:00 – 21:00 (IST)
  - 03:30–06:30 ; 08:30–10:30 ; 12:30–15:30 (UTC)
- Language: English
- Mentor: Stefan Blumentrath

Project Proposal

Project Title

Enhancing searchable metadata in Space Time Datasets for GRASS Temporal Framework.

Abstract

The temporal framework in GRASS was initially developed almost 15 years ago. Spatio-temporal data has become more widely available since then and new standards emerged like e.g. *STAC* with *extensions*, or the *CF-conventions*.

The aim of this project is to extend the metadata model in the temporal framework to improve discoverability of map datasets registered in TGIS and to allow users to embed extended metadata with the registered maps in a Space Time Dataset in a searchable way. Ideally, the new model is able to account for the standards mentioned above.

Problem Statement

The current TGIS metadata system in GRASS-GIS is limited in both structure and usability.

- metadata is not stored in a flexible or extensible format.
- it is not easily searchable or queryable
- it does not align with widely adopted standards such as STAC or CF-Conventions.

As a result users cannot filter datasets based on metadata attributes (e.g., cloud cover, sensor type, etc). This makes integration with external datasets difficult, also … discoverability of datasets within space time datasets is limited.

Additionally, the query mechanism `temporal_where` only supports time-based filtering and lacks the ability to incorporate metadata-based conditions. This creates a significant limitation in geospatial workflows, where metadata-driven filtering is essential.

Previous Contributions to GRASS

- **#6667 (merged):** adds docker labels from metadata
- **#7128 (merged):** makes periodic-updates use different branch for main and backport
- **#7151 (open):** re-wrote r.geomorphon JSON output to use parson library
- **#7211 (open):** prohibit writing map timestamps across mapsets

Schedule

Milestones

By the end of summer 2026, we want to achieve the following:

1. Updated TGIS database schema (v4) supporting structured metadata
2. Metadata stored in a flexible JSON-based format
3. Ability to query datasets using metadata conditions via `temporal_where`
4. Extended `t.create` and `t.register` to support metadata input
5. Migration tool for updating databases (v3 `->` v4)
6. New tool for updating and modifying metadata

7. Compatibility with universal metadata standards (STAC, CF-Conventions)

Investigation Plan and Research

To understand how metadata can be integrated into the temporal framework, I plan to investigate the system by answering the following:

- where and how is metadata currently stored in TGIS?
- how datasets are registered and what metadata is available at registration time?
- how does the `temporal_where` query mechanism operate internally?
- what are the limitations in extending the current schema?

A potential approach I can articulate would be:

1. analyze the current database schema and identify limitations in metadata storage.
2. study how metadata is passed through tools like `t.create` and `t.register`
3. investigate how queries are constructed and executed in TGIS
4. identify integration points where metadata-based filtering can be introduced.

From whatever I could infer during my time in the repository, I found a few inconsistencies in `r.info` and `r.timestamp` , only one of them shows timestamps, when maps are registered from a different mapset. Exploring these inconsistencies will be valuable in understanding how metadata is handled in raster and vector frameworks. Hopefully I can extend that understanding to the temporal framework as well.

I also noticed that SQLite is used as the backend, which supports JSON through the JSON1 extension.

This project proposes leveraging SQLite's JSON capabilities to store metadata and enable queries using functions, like the following:

```
json_extract(metadata, '$.cloud_cover') < 20
```

Implementation Plan & Timeline

April 1st - 19th: Preliminary Investigation and Design

- understanding the codebase and the exact pain-point at hand
    - analyze the current TGIS architecture, focusing on metadata storage and handling
    - identify limitations in the existing schema and query mechanisms
    - answer key questions such as: how are metadata parsed? and where are they stored?
        - basically trace the entire pipeline/ lifecycle of metadata in the temporal framework
    - propose a JSON-based metadata model, subject to validation after analysis
        - it should support both— standardized fields as well as user-defined extensions… without requiring future schema modification
    - document everything for future reference
- review relevant metadata standards such as STAC and CF-Conventions
- update mentors on a bi-weekly basis or whenever I have something concrete

I would like to inform the mentor(s) that I won't be available between 20th April 2026 - 9th May 2026, due to my end-sem exams being held then. 10th May onwards, I'm available for contributing to the project during the time slots specified above.

May 11th - 25th: Database Extension

- community bonding
- study integration points within tools such as `t.create`, `t.register` and `temporal_where`
    - possibly also explore inconsistencies such as `r.info` and `r.timestamp` if time permits
- extend the SQLite schema to include a metadata field (likely JSON-based), depending on feasibility
    - prototype the schema and test basic insert/read operations using sample metadata such as:

```
{
        "sensor": "Sentinel-2",
        "cloud_cover": 12,
```

```
        "resolution" : "10m"
    }
```

  - integrate the JSON storage using SQLite JSON1 functions for querying
- ensure backward compatibility
  - the implementation should ensure TGIS v3 still works, and the existing tools don't break.
  - mainly aim to preserve workflows wherever possible; core idea is that users shouldn't be forced to change their workflows.

May 26th - June 10th: Tool Integration

- work on input handling
  - extend `t.create` to accept dataset-level metadata (JSON input)
    - perform basic validation (like JSON parsing) before storing in the new schema
  - extend `t.register` to accept map-level metadata
    - attach metadata to each map entry… something like

```
t.register input=dataset maps=map1 metadata='{"cloud_cover": 15}'
```

syntax is subject to refinement based on existing CLI conventions

- backend connection and compatibility
  - pass metadata from CLI `->` Python TGIS `->` SQLite backend
  - if metadata field is NULL, queries should ignore metadata conditions
- modify the documentation to show the changes made in `t.create.md` and `t.register.md`

June 11th - 25th: Query System Enhancement

- extend `temporal_where` syntax to support metadata filters:
  - an example of how that would look:

currently: `temporal_where="start_time >= '2020-01-01'"`

modified: `temporal_where="start_time >= '2020-01-01' AND metadata.cloud_cover < 20"`

- extend the existing query parsing mechanism to interpret metadata conditions and translate them into SQLite JSON queries (e.g. using json_extract)
  - initial support will focus on simple conditions, with incremental support for more complex expressions.

June 26th - July 10th: Migration Tool

The migration tool will be responsible for upgrading existing database from v3 to v4… ensuring there is no data loss (risks and mitigation strategies have been discussed below)

- **Migration Strategy**
  the migration will aim to be non-destructive, preventing data loss while introducing the new schema
    - detect current schema (v3)
    - create an updated schema (v4) with updated metadata fields
    - migrate existing records into the new structure

- migration approach will be refined after analyzing existing database structures and edge cases during implementation

July 11th - 20th: Metadata Editing Tool

- implement CLI tool for updating metadata
- support updating and extending metadata entries

July 20th - August 1st: Testing and Final Documentation

- unit tests for metadata storage and querying
- integration tests with TGIS workflows
- documentation and usage examples

The proposed metadata system will be designed to remain extensible, allowing future integration of evolving metadata standards without requiring major schema redesign.

Details on the implementation may evolve after deeper investigation during the community bonding phase.

Risks and Migitation

A project at this scale is bound to have some risks involved when handling large datasets. Here are some that I can think of:

1. SQLite JSON Performance
    a. Risk: JSON queries may be slower for large datasets
    b. Migitation:
        i. use indexing strategies if needed
        - instead of parsing thru the entire JSON blob, we could use indexing, SQLite lets us create an index on an expression

        ```
        CREATE INDEX idx_cloud_cover
        ON <some_table> (json_extract(metadata, '$.cloud_cover'));
        ```

        - Indexes will be introduced selectively for common queries. Usually the ones that are slow and take up time.. using logs/testing. Some common ones to start with could be cloud_cover, sensor etc.

        This avoids unnecessary indexing overhead, and drastically improves the performance for common queries. The idea is to start small and optimize based on observations.

        ii. benchmark the performance during Database Extension phase

2. Schema Migration Issues
    a. Risk: data loss or incompatibility during v3 `->` v4 migration
    b. Mitigation:
        i. develop a migration tool with fallback option
        ii. preserve original schema as backup
        - preserve original database state by creating a backup copy before migration (eg: duplicating the SQLite file), ensuring rollback in case of any failure

3. Query Parsing Complexity
    a. Risk: extending `temporal_where` may introduce parsing bugs
    b. Mitigation:
        i. incremental tests, with simple queries, and then move to complex queries.
        ii. add unit tests for parsing logic.

Commitments during the Summer

I have no significant commitments during this summer, apart from a mandatory internship that I have to pursue, as part of my course evaluation. I'll be staying back on campus during the summer, and pursue the internship under a faculty member at campus.

The research project won't be intellectually stimulating either, it's more of a formality at this point. I've ensured that is the case, to make my workload as trivial as possible. The typical work hours I'll have to spend towards the internship is around 8-10 hours per week at max, leaving me with about 30-40 hours per week to commit to OSGeo.

As mentioned above, I won't be available between 20th April – 9th May due to my end-sems. I'm available during the remainder of the coding period.

Should there be any unforeseen circumstances, I will inform the mentors beforehand. **I'm committed to finishing this project by the end of summer 2026, and I'd love to even continue maintaining it afterwards.**