UNIVERSIDADE
BEIRA INTERIOR

Optimization of GNSS Data Archival and Exchange using TileDB

Duarte Nuno de Brito Arribas

Dissertação para obtenção de grau de mestre em Engenharia Informática (2º ciclo de estudos)

Orientador: Rui Manuel da Silva Fernandes
Coorientador: Paul Andrew Crocker

Covilhã, janeiro de 2025

Declaração de Integridade

Eu, Duarte Arribas, que abaixo assino, estudante com o número de inscrição M12324 do Mestrado em Engenharia Informática do Departamento de Informática, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o Código de Integridades da Universidade da Beira Interior.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 31/01/2025

Assinado por: **Duarte Nuno de Brito Arribas**
Num. de Identificação: 30693649
Data: 2025.02.03 12:09:45 +0000

Acknowledgments

The culmination of this dissertation and report would have not been feasible without the assistance of my advisors Rui Fernandes, Ph.D., and Paul Crocker, Ph.D., who believed and guided me with patience as I (mis)implemented their (and my) applications.
I would like to express my sincere gratitude to the following individuals, whose support and assistance were crucial in the successful completion of this dissertation and my academic journey:

- To all the SEGAL and MiraSpaco staff, collaborators, and researchers, especially Fernando Geraldes, Diogo Ventura, and Luís Carvalho, for their invaluable help in overcoming challenges and providing the necessary material for my research;
- To my parents and family, whose unwavering belief in me created a comfortable and supportive learning environment throughout the years. Their encouragement and assistance helped me navigate difficult times and stay resilient in the face of adversity;
- To my university colleagues, who motivated me during the most stressful periods, never letting me give up;
- To my friends, who provided much-needed relaxation and distraction during times of stress, making sure I remained balanced;
- And to everyone else who, both directly and indirectly, contributed to my success in any way;
- And to the "undo" button, for saving me from countless mistakes.

Resumo

O crescente volume de dados *GNSS* apresenta desafios significativos no seu armazenamento, processamento e interoperabilidade, especialmente devido às ineficiências do formato *RINEX*, que, embora amplamente utilizado, tem dificuldades em lidar com grandes volumes de dados e não é otimizado para ambientes em *cloud*. Esta dissertação explora o potencial das bases de dados baseadas em *arrays*, em particular o *TileDB*, como uma solução moderna para otimizar a gestão de dados *GNSS*. Uma contribuição central deste trabalho é o desenvolvimento de duas aplicações: `raw2rin` e `tiledb_manager`, que facilitam a conversão de ficheiros em formatos proprietários de recetores *GNSS* para os formatos *RINEX* e *TileDB*, apoiando a tradução bidirecional entre os últimos, oferecendo ajuste de parâmetros, correções de metadados, e o armazenamento e consulta eficientes de dados. A dissertação também inclui uma avaliação do desempenho do *TileDB*, focando-se nos efeitos de diferentes granularidades de dados e fragmentação. Os testes realizados apontam para que o desempenho de leitura poderá ser prejudicado à medida que a quantidade de dados e fragmentos aumentam.

Palavras-chave

TileDB, *GNSS*, *RINEX*, Armazenamento de Dados, Armazenamento pronto para análise e otimizado para a *cloud*, Arquivo de Dados, Intercâmbio de Dados.

Resumo alargado

Nos últimos anos, o uso de sistemas *GNSS* tornou-se fundamental em várias áreas, como geodesia, navegação, e monitorização ambiental, gerando uma quantidade crescente de dados que precisam de ser armazenados e processados eficientemente. O formato *RINEX*, que é o formato *standard* utilizado para a troca de dados *GNSS*, embora seja uma solução consolidada, apresenta limitações notáveis quando se lida com grandes volumes de dados. A sua estrutura de armazenamento linear e a falta de otimização para ambientes em nuvem tornam-no ineficiente quando se trata de gerir dados em larga escala. Este trabalho tem como um dos objetivos explorar formas de superar essas limitações, utilizando bases de dados baseadas em *array* otimizadas para grandes volumes de dados, como o *TileDB*.

O *TileDB* permite o armazenamento e manipulação eficiente de dados esparsos e multidimensionais, sendo, portanto, uma excelente solução para armazenar dados *GNSS*. O objetivo central da dissertação é desenvolver um sistema que facilite a conversão e o processamento desses dados eficientemente. Para isso, foram desenvolvidas duas ferramentas essenciais: `raw2rin`, uma aplicação que converte os dados dos recetores *GNSS*, que frequentemente utilizam formatos proprietários, para o formato RINEX, que permite, ainda, a alteração de parâmetros e correções de metadados, e `tiledb_manager`, que converte esses dados para *TileDB* e de volta para *RINEX*.

Além das aplicações desenvolvidas, o trabalho inclui uma análise do desempenho do *TileDB*. Mediante uma série de testes, foi possível avaliar o impacto de diferentes níveis de granularidade dos dados e o efeito da fragmentação na desempenho de leitura das bases de dados. Os resultados indicam que, à medida que o número de fragmentos de uma TileDB ou quantidade de dados pedidos aumenta, o desempenho de leitura do *TileDB* degrada-se.

Abstract

The growing volume of GNSS data presents significant challenges in its storage, processing, and interoperability, particularly due to the inefficiencies of the RINEX format, which, while widely used, struggles to handle large datasets and is not optimized for cloud environments. This dissertation explores the potential of array-based databases, particularly TileDB, as a modern solution to optimize GNSS data management. A central contribution of this work is the development of two applications: `raw2rin` and `tiledb_manager`, which facilitate the conversion of files of proprietary GNSS receiver formats to RINEX and TileDB formats, supporting bidirectional translation between the latter, parameter adjustment, metadata corrections, and efficient data storage and querying. The dissertation also includes an evaluation of TileDB's performance, focusing on the effects of different data granularities and fragmentation. The tests conducted point to a worse read performance for both higher data retrieved, and higher fragment numbers.

Keywords

TileDB, GNSS, RINEX, Data Storage, Analysis-Ready and Cloud-Optimized Storage, Data Archiving, Data Exchanging.

Contents

List of Figures

List of Tables

Acronyms

ACM association for computing machinery

AJAX asynchronous JavaScript and XML

API application programming interface

ARCO analysis-ready and cloud-optimized

ASCII american standard code for information interchange

AUSPOS australian positioning online service

AWS amazon web services

CCS computing classification system

CLI command-line interface

CNB ComNav binary format

CPU central processing unit

CRUX conversion and update of RINEX files

CSRF cross-site request forgery

CSRS-PPP canadian spatial reference system - precise point positioning

CSV comma-separated values

EPOS-GNSS european plate observing system - global navigation satellite system

ER entity relationship

FCN frequency channel number

GCM Galois/Counter mode

GSI GEO serial interface

GFZ geoForschungsZentrum

GLONASS globalnaya navigazionnaya sputnikovaya sistema

GNSS global navigation satellite system

GPS global positioning system

HDF5 hierarchical data format version 5

HTML hypertext markup language

HTTP hypertext transfer protocol

I/O	input/output
ID	identification number
IGS	international GNSS system
IRNSS	indian regional navigation satellite system
JSON	javaScript object notation
LLI	loss of lock indicator
MBR	minimum bounding rectangle
MEng	master of engineering
NavIC	navigation with indian constellation
OS	operating system
PNT	positioning, navigation, and timing
PT	propagation time
QZSS	quasi-zenith satellite system
REST	rEpresentational State Transfer
RFC	Request for Comment
RINEX	receiver INdependent EXchange
RSS	resident set size
S3	simple storage service
SATNAV	satellite navigation
SBAS	satellite based augmentation system
SBF	septentrio binary format
SEGAL	space and earth geodetic analysis laboratory
SNR	signal-to-noise ratio
SQL	structured query language
SQS	simple queue service
SNS	simple notification service
SSH	secure shell
SSI	signal strength indicator

STH	SOUTH
UBI	university of beira interior
UNAVCO	university NAVSTAR Consortium
URL	uniform resource locator

Chapter 1

Introduction

1.1 Scope and Motivation

Satellite-based navigation systems have become indispensable tools in the contemporary days. GNSS is a key technology, encompassing satellite constellations like GPS and GLONASS, which provides vital data for navigation, positioning, and timing, playing a crucial role in global communication, transportation, and emergency services.

On account of GNSS systems generating abundant measures of data, there is an ongoing demand for advanced data storage and management solutions. Distinct receivers' manufacturers generate GNSS data in varying, sometimes proprietary and closed-source, formats, aggravating interoperability between communities and services. Adopting standards like RINEX ensures consistent data storage, facilitates compatibility across different GNSS systems, and simplifies the processing and exchange of data among interested parties.

Within the academic context of the *Engenharia Informática* (9286) MEng course at the University of *Beira Interior* (UBI), this document comprises the investigation study of the dissertation in the scope of the curricular unit of *Dissertação em Engenharia Informática* (14477). The goal is to summarize the research conducted, delineate the system that was built, present the methodologies employed, and analyze the results obtained to demonstrate the contributions and insights of this work.

The scope of this work lies at the intersection of the areas of **Data Storage** and **Geoscience**. Given the nature of the dissertation's focus on data storage solutions for GNSS data, databases and application programming interfaces (APIs) stand out as crucial components for the successful execution of research and development tasks.

Under the 2012 version of the association for computing machinery (ACM)'s computing classification system (CCS)[ACM], the de facto standard classification system for the computing fields for computer science, the areas that encompass this dissertation are:

- **Database web servers**;
- **Application servers**;
- **Earth and atmospheric sciences**;
- *Cloud based storage*;
- *Multidimensional range search*;
- *Incomplete data*;
- Data compression;
- Data encryption.

1.2 Problem Statement

The two main problems addressed in this work are:

- Multiple proprietary receiver formats hinder processing;
- RINEX's inefficiency on maintaining an archive.

1.2.1 Multiple Proprietary Receiver Formats Hinder Processing

Because GNSS data is often provided in proprietary RAW file formats unique to each equipment manufacturer, it is often difficult and inaccessible for users to process and analyze data efficiently, as each format requires specific software tools that are often platform-dependent and not interoperable.
Currently, there is no unified system capable of handling multiple proprietary formats that ensure seamless conversion and metadata correction to RINEX.

1.2.2 RINEX's Inefficiency on Maintaining an Archive

The RINEX files' text-based nature introduces challenges such as during data retrieval, as RINEX files must be parsed using an iterative line reading procedure, which causes significant overhead, particularly for lengthy files.
The division of RINEX files into data and metadata portions introduces additional challenges, as changes in metadata require updates across all files, potentially resulting in redundancy and inconsistency. Furthermore, the practice of staging files with data before metadata arrival and verification is common due to their asynchronous arrival.
While RINEX version 4.01 is the latest, many processing software and entities continue to work with previous versions. Generating new versions or sample rates from the original RINEX files proves to be computationally expensive, and maintaining multiple copies under different versions or sample rates occupies a significant amount of space.
Due to the nature of files, they are not analysis-ready and cloud-optimized (ARCO), especially since it is cost prohibitive to upload the current abundance of files (in the case of several GNSS communities) to the cloud.

1.3 Objectives

Addressing the challenges mentioned in Section 1.2.2 is crucial to establish an efficient and reliable system for storing and retrieving GNSS data. Thus, this dissertation's central objectives are (i) to implement a conversion framework, by creating an application that translates receiver-specific formats to RINEX, which also establishes a bidirectional translation between it and the novel TileDB methodology, and (ii) to evaluate TileDB as the novel storage methodology for GNSS data, positioned as a contender to the ubiquitous universal format, RINEX, whose read and write performance will be evaluated through series of tests to systematically ascertain its possible advantages.

1.4 Report Outline

The sequential design of this document ensures a coherent and clear reading experience, meaning each chapter is meticulously written to maintain clarity and logical continuity; thus, this text was thoroughly grouped into the following six chapters:

Chapter 2 - Important concepts Delineates the core concepts of GNSS, RINEX and TileDB;

Chapter 3 - State-of-the-Art Explores the cutting-edge technology that is already available for GNSS data storage, including both UNAVCO and GFZ's current research approach on using TileDB to store GNSS data;

Chapter 4 - Raw2Rin Software Package Details the design, functionality, and implementation of the `raw2rin` software package for converting proprietary RAW GNSS files to the standardized RINEX format;

Chapter 5 - TileDB Data Generation and Storage Using TileDB Manager Explains the process of generating GNSS importable data for TileDB and managing it using the `tiledb_manager` application, including data import/export and metadata handling;

Chapter 6 - TileDB Evaluation Evaluates TileDB's performance and suitability for GNSS data storage through benchmarking and analysis of its storage and query efficiency;

Chapter 7 - Conclusion Summarizes the work conducted, highlights its contributions, and suggests potential directions for future research and development.

Chapter 2

Fundamental Concepts

2.1 Introduction

This chapter comprises the research and analysis of the fundamental concepts needed for this dissertation. First, Section 2.2 explores the core aspects of GNSS and the kind of data it generates. Section 2.3 explores the different receiver manufacturers and the kind of proprietary formats they include. Moreover, Section 2.4 further dives into the universal format for storing GNSS data, RINEX, its characteristics and structure. Finally, Section 2.5 presents TileDB as a possible contender to RINEX as a GNSS data archival system.

2.2 GNSS

Navigation is the discipline centered on guiding vehicles or individuals between locations. Typically, electronic-based systems are employed to facilitate accurate navigation; most notably, *satellite navigation* (SATNAV) systems are sophisticated networks of satellites, which provide data used to generate geospatial positioning, navigation, and timing (PNT)[Tho21]. Upon achieving global coverage, this system is conferred with the designation of *global navigation satellite system* (GNSS)[Gov21], meaning it embodies all SATNAV systems and their augmentations and provides accurate, continuous, worldwide, three-dimensional position and velocity information to users with the appropriate receiving equipment[KH17].
At the time of writing, there are six core GNSSs in the world[Rut] owned by different political organizations. Table 2.1 displays who owns each GNSS, the year each started their operation, and how many satellites they currently have in orbit.

GNSS name	Location	Operation start	Satellites in orbit
GPS	USA	1978	31[Gov24, Gov]
GLONASS	Russia	1993	26[IAC]
BeiDou	China	2000	45[Nova]
QZSS	Japan	2010	4[Novb]
IRNSS	India	2013	7[Orga, Orgb]
Galileo	EU	2016	24[Cen, Age]

Table 2.1: Details of the six core GNSSs in the world.

GNSS comprises three segments[Ash20, Tec]:

- *Space segment*: constellation of satellites equipped with atomic clocks deployed in almost circular orbits at approximately 20,000 kilometers from Earth, which provides their identity, time, orbit, and status;
- *Control segment*: A network of ground stations, comprising receivers and antennas, which is established to conduct monitoring activities for the satellite constellation. It

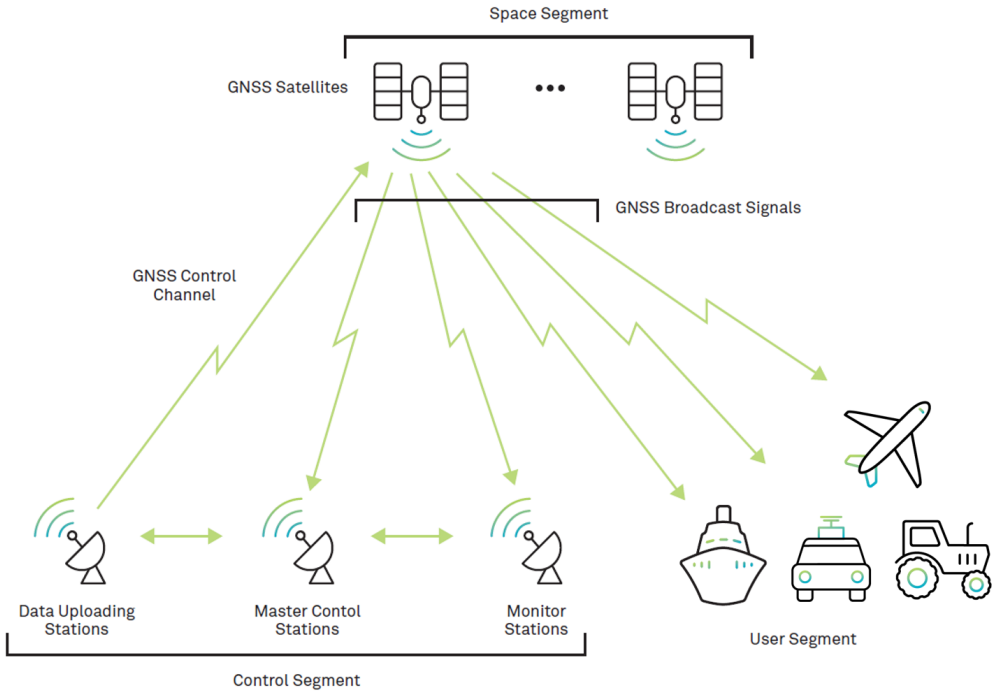


Figure 2.1: GNSS segments. From NovAtel Inc. in "An Introduction to GNSS", 2015[Inc15].

encompasses an array of ground-based monitoring stations, which observe the signals and status of the satellites, relaying them to the master control stations, which, in turn, implement corrections to the satellites' orbits, so that they accurately reach the data uploading stations;

- *User segment*: instruments with GNSS receivers that track GNSS signals and produce position and velocity solutions, typically with low-cost clocks.

Figure 2.1 shows how each segment relates to each other.
The satellites contain very accurate (atomic) clocks[Inc15] and transmit their data to an *antenna*, which captures electromagnetic waves and converts them into electrical signals for a receiver to process[MO09]. A *receiver* is a specialized device designed for processing the previously received signals and delivering positioning data[1] using advanced chipsets and algorithms to track and generate accurate navigation information[POL+08, SEPb].
The positioning data is calculated by using ranges and specific algorithms. Consider the space vector Φ^S with respect to the center of the Earth for each satellite, which can be computed from the satellite broadcast of navigation data (ephemerides), and the space vector Φ^r relative to the center of the Earth for each receiver, represented in Figure 2.2.
The geometric distance Φ can be accurately measured by recording the run time required for the satellite signal to reach the receiver, as shown in Equation 2.1.

$$\Phi = ||\Phi^S - \Phi^r|| \quad (2.1)$$

[1]Such as, for example, latitude, longitude, and height.

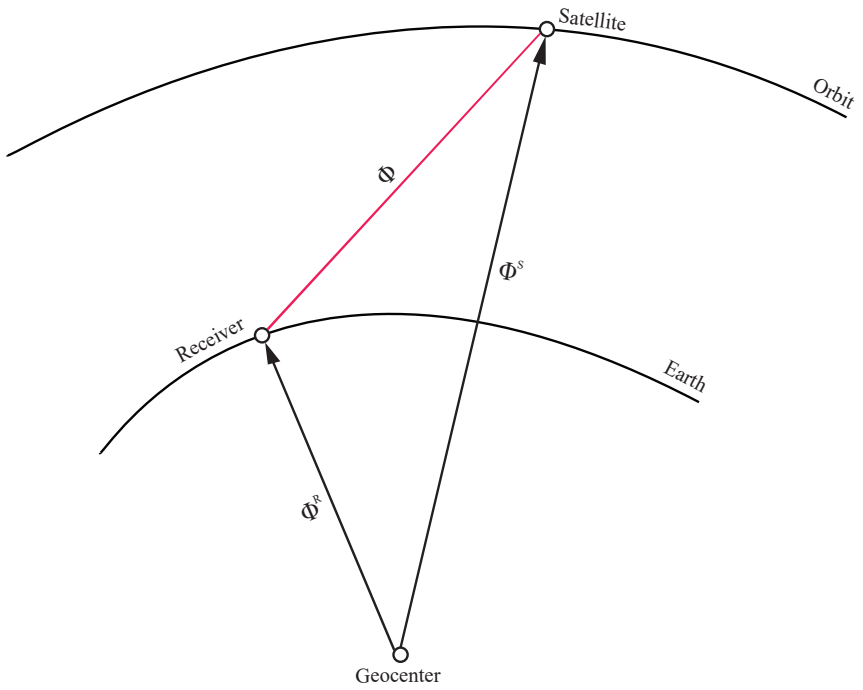


Figure 2.2: Space vectors of satellites and receivers with relation to the Earth.

In an idealized context devoid of errors, the geometric distance can be calculated with precision. However, natural errors[Inc15], including inaccuracies arising from the refraction of satellite signals as they traverse the ionosphere and troposphere, as well as equipment errors[err] often stemming from the suboptimal quality of the equipment involved in the GNSS data collection process, introduce uncertainties. Consequently, the accurate determination of range using geometric distance becomes impractical[PHR+23].

Instead, a new measurement, termed *pseudorange* ($\tilde{r}$) is employed, which represents the distance between the GNSS satellite at the instant of signal emission and the receiver antenna, measured at the time of reception. Equation 2.2 outlines the procedure for calculating the pseudorange[Rom21].

$$\tilde{r} = \Phi + c(r - s) + \text{other biases} \quad (2.2)$$

, where c is the speed of light, r is the receiver clock offset and s is the satellite clock offset. Each range corresponds to the surface of a sphere centered at the respective satellite. The receiver's position can be accurately estimated through the application of the *trilateration* principle, a method that refines the potential locations of the receiver. This process necessitates information on distances and orbits from a minimum of four satellites. Figure 2.3 provides a visual representation elucidating the trilateration process.

In this process, the orange satellite determines the receiver's location to lie somewhere on a sphere. The gray satellite refines the potential receiver locations to the intersection circle between the two spheres. Subsequently, the blue satellite further narrows down the possibilities to two points and, finally, the green satellite precisely pinpoints the receiver's exact position.

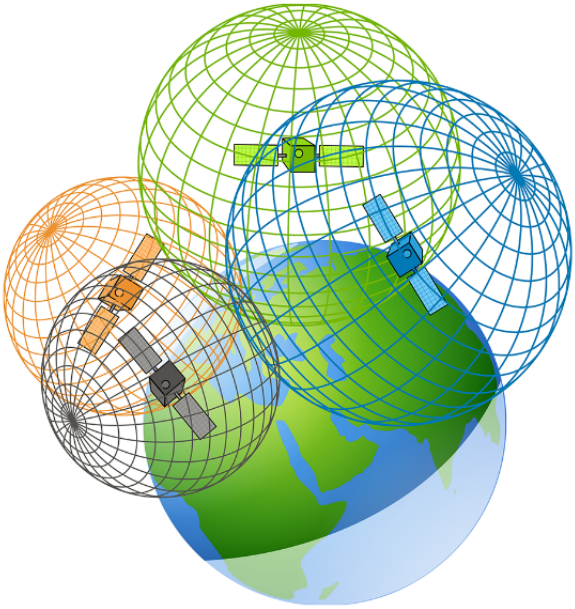


Figure 2.3: Four satellites used for trilateration around a location on Earth. Adapted from GISGeography in "How GPS Receivers Work – Trilateration vs. Triangulation", 2022[GIS].

2.3 Proprietary Formats

The receivers are producer-dependent, hence each adopts its own architecture and algorithms. One important aspect is how each stores the GNSS data, as each secures its own proprietary format. Table 2.2 lists some common GNSS receiver producers and their proprietary formats.

Receiver Manufacturer	**Data Format**
Septentrio[SEPa]	Septentrio binary format (SBF) (`.sbf` file)[SBF]
Trimble[TRI]	DAT or T02/T04 formats (`.dat` or `.t02`/`.t04` files)[DAT]
Leica Geosystems[LEI]	GEO serial interface (GSI) format (`.gsi` files) or M00 (`.m00` files) [GSI, GSI03, sur15]
ComNav Technology[COMb]	ComNav binary format (CNB) (`.cnb` files)[CNB]
South Surveying[SOU]	SOUTH (STH) format (`.sth` files)[STH]

Table 2.2: Most popular GNSS receiver manufacturers.

Because each format is different from each other, there was a need to introduce an industry-standard format for storing GNSS data.

2.4 RINEX

The *Receiver INdependent EXchange* (RINEX) file format was developed by the Astronomical Institute of the University of Bern for the easy exchange of data from one specific type of GNSS[2], the GPS[Rom21].
It is the universal standard for storing and exchanging GNSS observation, navigation, and meteorological data and seeks to endorse interoperability among the dissimilar receiver brands

[2]Later expanded to the remaining popular GNSSs.

and processing software. Ergo, it is of particular relevance that its details are well-defined and documented, as several GNSS communities, such as the *australian positioning online service* (AUSPOS), the *canadian spatial reference system - precise point positioning* (CSRS-PPP), the *european plate observing system - global navigation satellite system* (EPOS-GNSS), and others, rely on user-submitted data in compliance with the RINEX standard for their operations[Jan23]. The whole format is well-documented in a series of Request for Comments (RFCs), pertaining to each RINEX version. It consists not only of one file type, but instead three ASCII file types, which store their respective data[GHH+09]:

- *Observation data files*: store observation ("Obs") data, viz., (pseudo)range, phase, Doppler, or other signal quantity computed with a GNSS receiver;
- *Navigation data files*: store navigation ("Nav") data, viz., broadcast of satellite navigation data records;
- *Meteorological data files*: store meteorological ("Met") data.

Each observation and meteorological data files usually[3] contain the data from one site and session only. Amidst this dissertation, the primary focus will be on RINEX observation files, given their widespread usage and the substantial volume of data they encompass.

2.4.1 RINEX File Naming Conventions

RINEX filenames are of uttermost importance, as they reveal crucial information about the contents of the files. Figures 2.4[GE07a] and 2.5[GE07b] represent the filenames across versions 3.01 (and previous) and 3.02+, respectively.

```
ssssdddf.yyt

|   |  | | |
|   |  | | +--  t:  file type:
|   |  | |          O: Observation file
|   |  | |          N: GPS navigation message file
|   |  | |          M: Meteorological data file
|   |  | |          G: GLONASS navigation message file
|   |  | |          L: Galileo navigation message file
|   |  | |          P: Mixed GNSS navigation message file
|   |  | |          H: SBAS Payload navigation message file
|   |  | |          B: SBAS broadcast data file
|   |  | |                      (separate documentation)
|   |  | |          C: Clock file (separate documentation)
|   |  | |          S: Summary file (used e.g., by IGS, not a standard!)
|   |  | +---  yy:  two-digit year
|   |  +-----   f:  file sequence number/character within day.
|   |               daily file: f = 0 (zero)
|   |               hourly files:
|   |               a =  1st hour: 00h-01h; b =  2nd hour: 01h-02h;
|   |                           . . .       x = 24th hour: 23h-24h
|   +-------  ddd:  day of the year of first record
+-----------  ssss:  4-character station name designator
```

Figure 2.4: (≤) 3.01 RINEX filename description.

The filenames of versions 3.02 and onward are larger and contain a greater level of detail than those of previous versions.

[3]Technically, the format supports more than one receiver per file, but the RFC does not recommend it even if their time does not overlap.

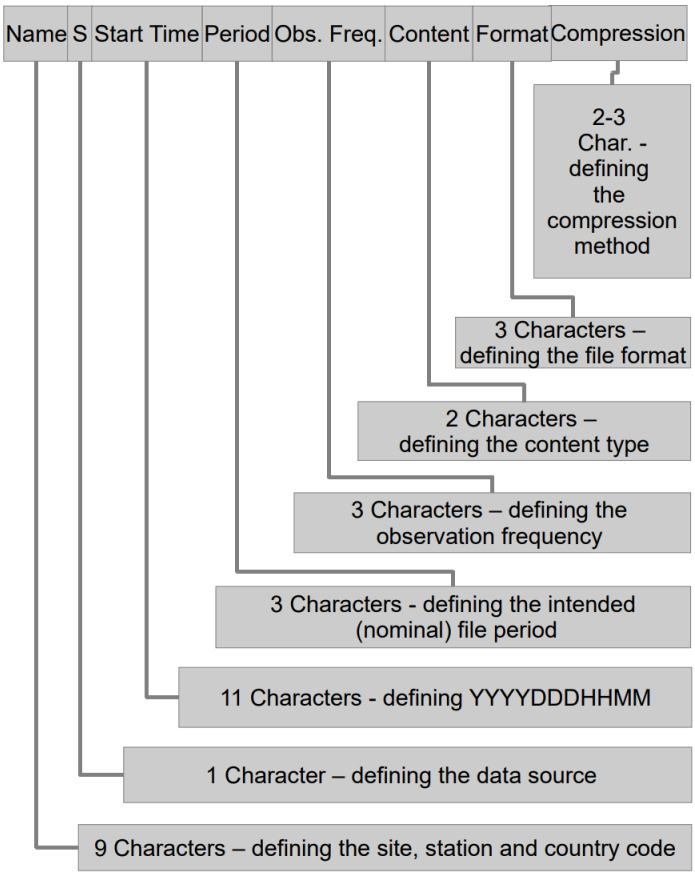


Figure 2.5: (≥) 3.02 RINEX filename description.

2.4.2 RINEX File Structure

Each file type is structurally composed of two segments: the header (metadata) section and the data section.

2.4.2.1 Header (metadata) Section

Positioned at the file's beginning, the *header section* is divided into header lines, each one entailing file-wide information that shall be used in post-processing applications[LAN] by establishing coherence to the data section.
The metadata exhibits differences across distinct file types, i.e., its possible fields within an observation file diverge from those observed in a navigation file; however, they present an akin layout: each line must contain no more than 80 characters, reserving characters 61-80 for a mandatory *header label*, indicating what type of information the preceding *line segment* (1-60) refers to[4]. Listing 2.1 represents the metadata of a RINEX 3 observation file.

Listing 2.1: A RINEX 3 observation file header.

```
5    10   15   20   25   30   35   40   45   50   55   60   65   70   75   80
```

[4]Note that both the header label and header information segments may not be entirely filled, resulting in the utilization of blank spaces to occupy the remaining character spaces. This practice ensures conformity to the stipulated structure, allowing flexibility in accommodating varying lengths of header labels and associated information.

```
     3.05           OBSERVATION DATA    M (MIXED)           RINEX VERSION / TYPE
sbf2rin-15.6.1                          20231121 123841 UTC PGM / RUN BY / DATE
gfzrnx-2.1.0        FILE PROCESSING     20231121 164627 UTC COMMENT
WELL00GBR                                                   MARKER NAME
0                                                           MARKER NUMBER
CORS                                                        MARKER TYPE
C4G                 C4G                                     OBSERVER / AGENCY
3055312             3SNAVIGATION        3SNAV GNSS-300      REC # / TYPE / VERS
1910180014          3SNAVIGATION                            ANT # / TYPE
  4394607.8938 -2354743.6206  3965148.0973                  APPROX POSITION XYZ
        0.0000        0.0000        0.0000                  ANTENNA: DELTA H/E/N
C    6 C2I C6I C7I L2I L6I L7I                              SYS / # / OBS TYPES
E   10 C1C C5Q C6C C7Q C8Q L1C L5Q L6C L7Q L8Q              SYS / # / OBS TYPES
G   11 C1C C1L C1W C2L C2W C5Q L1C L1L L2L L2W L5Q         SYS / # / OBS TYPES
I    2 C5A L5A                                             SYS / # / OBS TYPES
J   10 C1C C1L C1Z C2L C5Q L1C L1L L1Z L2L L5Q             SYS / # / OBS TYPES
R    4 C1C C2C L1C L2C                                     SYS / # / OBS TYPES
S    4 C1C C5I L1C L5I                                     SYS / # / OBS TYPES
  2020    12    25    18     0    0.0000000     GPS        TIME OF FIRST OBS
  9 R01  1 R07  5 R08  6 R12 -1 R13 -2 R14 -7 R22 -3 R23  3 GLONASS SLOT / FRQ #
    R24  2                                                  GLONASS SLOT / FRQ #
                                                            END OF HEADER
```

The metadata in RINEX files includes both mandatory and optional labels. In Listing 2.1, exclusively the obligatory header labels are presented, excluding the `COMMENT` record, which serves to incorporate any additional non-standard information into the header.

Notably, the `RINEX VERSION / TYPE` record marks the start of the file, as it must, always, stand as the initial record in a file, followed by the `PGM / RUN BY / DATE` record and concluding with the `END OF HEADER` as the closing record in the header section. The sequence of the remaining header labels can be flexibly rearranged, and their descriptions and formatting guidelines are meticulously delineated in the RINEX RFCs[Rom21]. Each line's formatting rule is a string of the general form `[Repetition] type format`, where the *type format* is defined in Table 2.3[Pes15].

Type format	**Data Type**	**Examples**	**Description**

`In/In.m`	Integer	`I3 : 123`
		`I3.5 : 00123`
		`I3.5 : -0123`
		`I4 : \#123`
			An integer with `n` digits (including the sign). A floating point forces the `n` number to always be of `m` size, being padded with zeros, if its initial size does not match that of `m` digits.

`Fn.m`	Float	`F4.2 : 1.31` `F5.1 : -11.2`	A float with `n` digits (including the sign and floating point number) and `m` decimal places.

`Dn.m/En.m`	Power	`D7.1 :`
		`1.5D+02`
		`E8.1:`
		`-0.1E+06`
			A double exponential with `n` digits (including the sign, floating point number, exponential letter and the exponent itself) and `m` decimal places. Note that the letter of the type format is the letter used in the actual number; the reasoning for the acceptance of the two letters (`E` and `D`) is compiler compatibility.

`An`	String	`A3 : GPS`
		`A7 :`
		`\#String`
			A string with `n` digits.

`nX`	Blank	`1X : \#`
		`5X :`
		`\#\#\#\#\#`
			`n` blank characters.

Table 2.3: Header data formats for single variables. `#` is used to represent blanks.

The *repetition* is the number of times the *data type* repeats and commas separate several formats in the same line, which should be read one after the other. For example, the format `F9.2,11X,A1,19X,A1,19X` consists of a floating-point number of nine digits, where three are decimal points (with the dot included); then, it contains eleven spaces, a string of one character, nineteen spaces, another string of one character and another nineteen spaces.

2.4.2.2 Observation file data section

The data section, succeeding the header, functions as the repository for the content specific to the file type. The observation files contain two types of lines[PDMM16]:

- *Epoch Record*: line that includes the time (epoch) of the record. Always starts with a `>`;
- *Data Record*: line that starts with an `snn`, i.e., a satellite code identifying the constellation and its slot number (possible satellites and their codes are specified in Table 2.4). followed by the observation types for each satellite, which are defined in the `SYS / # / OBS TYPES` metadata header section of the RINEX file.

Code	Constellation
G	GPS
R	GLONASS
S	SBAS Payload
E	Galileo
C	BeiDou
J	QZSS
I	NavIC

Table 2.4: Possible `snn`s.

Listing 2.2 depicts the excerpt of the data section of an observation file.

Listing 2.2: A RINEX 3 observation file data section.

```
    5    10   15   20   25   30   35   40   45   50   55   60   65   70   75   80
1  > 2006 03 24 13 10 36.0000000 0 5 -0.123456789012
2  G06 23629347.915 .300 8 -.353 4 23629347.158 24.158
3  G09 20891534.648 -.120 9 -.358 6 20891545.292 38.123
4  G12 20607600.189 -.430 9 .394 5 20607600.848 35.234
5  E11 .324 8 .178 7
6  S20 38137559.506 335849.135 9
7  > 2006 03 24 13 10 54.0000000 0 7 -0.123456789210
8  G06 23619095.450 -53875.632 8 -41981.375 4 23619095.008 25.234
9  G09 20886075.667 -28688.027 9 -22354.535 7 20886076.101 42.231
10 G12 20611072.689 18247.789 9 14219.770 6 20611072.410 36.765
11 R21 21345678.576 12345.567 5
12 R22 22123456.789 23456.789 5
13 E11 65432.123 5 48861.586 7
14 S20 38137559.506 335849.135 9
```

The values listed after the `snn` correspond to various types of measurements, including pseudoranges, carrier phases, Doppler shifts, signal-to-noise ratios (SNRs), and cycle slips. These observations are provided for each satellite observed at that specific epoch.

The `SYS / # / OBS TYPES` line will list the observation types, and each satellite's data record will match the order of these types. For instance, if the header line lists the following observation types: `G / 4 / C1C L1C D1C S1C`, it means that it has four observation types for GPS systems and its data columns will contain the following measurements: `C1C` (code pseudo-range on the `L1` frequency), `L1C` (carrier phase on the `L1` frequency), `D1C` (doppler shift on the `L1` frequency), and `S1C` (signal-to-noise ratio on the `L1` frequency). A corresponding data record could look like: `G01 1234567890  100.1234  2.3456  12.6789`[Pes15].
Additional flags like loss of lock indicator (LLI) and signal strength indicator (SSI) may also be included and indicate if a satellite signal was lost temporarily during the observation period and the signal strength (a complement to SNR), respectively[Rom21].

2.4.2.3 Problems with RINEX

The RINEX file format is, by definition, text-based[TEX]. Text-based formats, although easy to work with, entail several disadvantages[Ber23] over other forms of storage, such as databases.
Data retrieval on files is time-limited. Software that processes RINEX may need to read only a small portion of the file, but still needs to read line-by-line until the needed portion is read. For compressed files (such as Hatanaka[Hat08] files), even if only a small portion of the file is needed, the whole file still needs to be decompressed to be read. This creates a significant overhead, especially because RINEX files may be thousands of lines long, as the data pertains to a specific period in a specific frequency for one site and session only[PDMM16].
The metadata portion contains some information[5], such as marker ID, GNSS antenna / receiver, approximate coordinates, measurement time, and more.[RIN]. Because both the data and metadata parts are stored inside a single file, several problems start to arise:

- Changing metadata implies that all metadata be changed as well, e.g., if a receiver name changes, all files containing that receiver name must be changed;
- Files need to be staged (placed in a temporary repository) before metadata arrives and is verified. Often, the data and metadata do not arrive at the storage site at the same time[Ber23], so the files need to be stored somewhere before it arrives and is verified.

Although the most recent version of RINEX is version 4.01[Tra22], most processing software and entities still work with previous versions, such as 2.11, 3.05 and others. Likewise, several distinctive samples rates for the same information may be desired. Generating new versions and/or samples rates from RINEX files is computationally expensive[Ber23] and maintaining several copies of the same file under different versions/samples rates takes up space and may result in data redundancy and inconsistency.

2.5 TileDB

TileDB is an array database designed for efficiently storing, querying and accessing array data, notably sparse array data. It is currently implemented for a single node, supports high

[5] Especially useful for the processing software.

performance linear algebra[L+15, Li17], and was devised by Stavros Papadopoulos et al. in [PDMM16].
It was initially conceived as a scientific data storage and management system, and it has proven particularly advantageous due to its capability to facilitate the efficient retrieval of specific relevant areas within the array[GCRS17].

2.5.1 Architecture Description

In TileDB, an array is composed of cells within an n-dimensional space. Each cell is uniquely identified by *coordinates*, defined as a set of values within a dimension's domain range and may be empty or contain a tuple of values, known as attributes. A *dimension* corresponds to a value space E^i, where i denotes a coordinate value constrained by a minimum and maximum value specified in its *domain*. On the other hand, an *attribute* is an atomic value characterized by a specific datatype, such as primitive types like integers, floats, or characters or fixed-/variable-sized vectors of primitive types, which can also be used to represent other complex data types, such as complex numbers or strings[BP17].
For example, in a 2D array (A), a tuple that identifies a set of three named attributes α is of type $A(i, j)$, as portrayed in Table 2.5.

	1	**2**	**3**	**4**
1	(14,red,2024-01-31)	(67,blue,2024-04-15)	(33,yellow,2024-07-21)	(46,white,2024-10-19)
2	(26,pink,2024-03-01)	(50,brown,2024-04-21)	(67,blue,2024-07-24)	(14,red,2024-10-30)
3	(78,blue,2024-03-23)	(27,green,2024-06-08)	(16,white,2024-08-20)	(78,yellow,2024-12-09)
4	(96,yellow,2024-04-06)	(37,red,2024-06-22)	(1,pink,2024-09-08)	(70,black,2024-12-15)

Table 2.5: A TileDB 2D 4×4 array with three attributes (Age, Favorite Color, and Next Birthday).

The domain is consistent for both dimensions, constraining them to a minimum value of 1 and a maximum value of 4. Therefore, each potential table query spans from $(1, 1)$ to $(4, 4)$. For instance, the query `A(2,2)["Favorite_Color"]` uniquely retrieves "brown".
In a 3D array, on the contrary, the tuple would extend to include a third dimension, denoted as $A(i, j, k)$, representing a cube.
Note that distinct attributes within the same array may even differ in datatype; for instance, in the case of the array represented in table 2.5, α_1 is an integer, α_2 is a string, and α_3 is a date.

2.5.2 Capabilities of TileDB

TileDB distinguishes itself as a capable storage management system, particularly excelling in handling large datasets.
The following subsections highlight the specific characteristics that make TileDB better-suited for storing substantial volumes of data than other comparable systems.

2.5.2.1 Dense Arrays and Sparse Arrays

There are two types of arrays in TileDB:

- *Dense array*: array in which every cell has an associated value, materialized in storage;

- *Sparse array*: array in which there may be array cells, which do not contain a value, such as "undefined" or "empty" values, which are not materialized in storage.

Figure 2.6 presents two Sub Figures, each with a logical view (more on this in Section 2.5.2.2) of two identical arrays with only the attribute α_1. The array of Sub Figure 2.6a is dense and the array of sub Figure 2.6b is sparse.
Note that the dense array contains all values filled up, including empty ones which are filled up with dummy values. In contrast, the sparse empty values are, indeed, empty and do not occupy storage space when persisted. The performance considerations of this will be discussed in Section 2.5.7.
The workflow of TileDB consists on:

- *Creating the array*: create the *array schema*[6] (the structure of the array), specifically the type of array (dense/sparse), the number of dimensions, their domain, the number, and type of attributes, the tile extent and cell/tile order (more on these two is covered in Section 2.5.2.2. The array is, then, stored as persistent storage in the form of directories on disk;

- *Writing to the array*: open the array for writing and emit the values needed for writing; by default, the writing is done on row-major order (but the remaining orders will be described in Section 2.5.2.2). Each attribute will be stored in a `.tdb` file with the name of the attribute. For instance, the values of an attribute named "color" will be stored in a file named `color.tdb`;

- *Reading from the array*: open the array for reading and read the values in the order defined previously by slicing the array.

2.5.2.2 Tiling

Attempting to access an attribute of a small subarray from a substantial, but compressed array, requires decompressing the entire `.tdb` file, which may be cost prohibitive. Furthermore, retrieving a cell at a time from the file is further exacerbated, as it may involve numerous input/output (I/O) system calls to the local file system or HTTP requests to a cloud system. To address these challenges, TileDB employs the concept of *tiling*, which involves the division of the array into chunks known as *tiles*, which hold a *tile extent* (i.e., a specific dimensional size). There are two distinct types of tiles: space tiles and data tiles. *Space tiles* represent a division of the array in its *logical view*, i.e., the actual n-dimensional array, while *data tiles* signify a segmentation in its *physical view*, meaning in the actual `.tdb` file.
Figure 2.7 depicts the logical view of a 2-dimensional untiled compressed array with dimensions 1000×1000 and only one attribute.
The red rectangle highlights the subarray to be read from the array. In this case, the `.tdb` file corresponding to the attribute α_1 must be fully decompressed, and each cell needs to be read until all six required cells are retrieved.

[6] Once created, the array schema cannot be changed.

cols

row

4	2	3	9	-2147483648	-2147483648
3	-2147483648	9	8	4	1
9	1	2	5	1	2
8	8	9	-2147483648	0	8
-2147483648	8	-2147483648	2	-2147483648	3
5	-2147483648	1	4	9	-2147483648

(a) An example of an unfilled dense array.

cols

row

4	2	3	9		
3		9	8	4	1
9	1	2	5	1	2
8	8	9		0	8
	8		2		3
5		1	4	9	

(b) An example of an unfilled sparse array.

Figure 2.6: Two equal unfilled TileDB arrays.

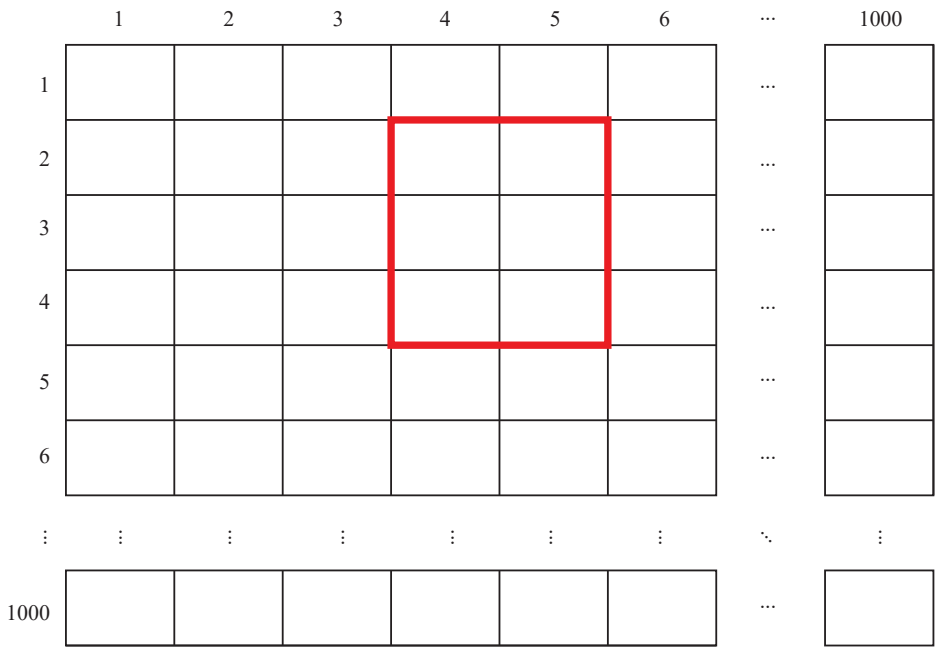


Figure 2.7: An untiled array.

On the contrary, Figure 2.8 represents the logical view of the same array subjected to tiling with a tile extent of 2×3.

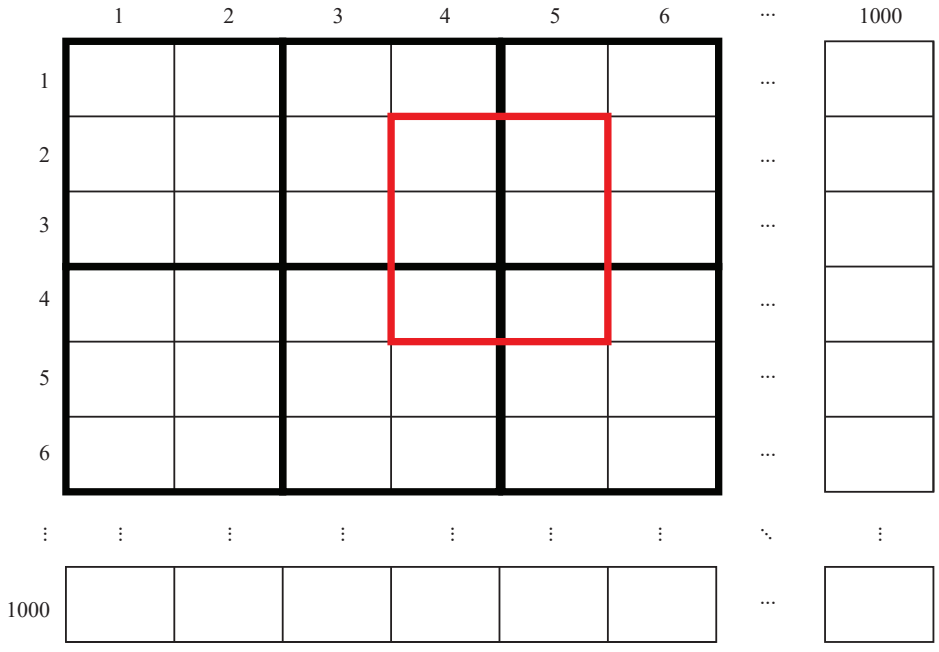
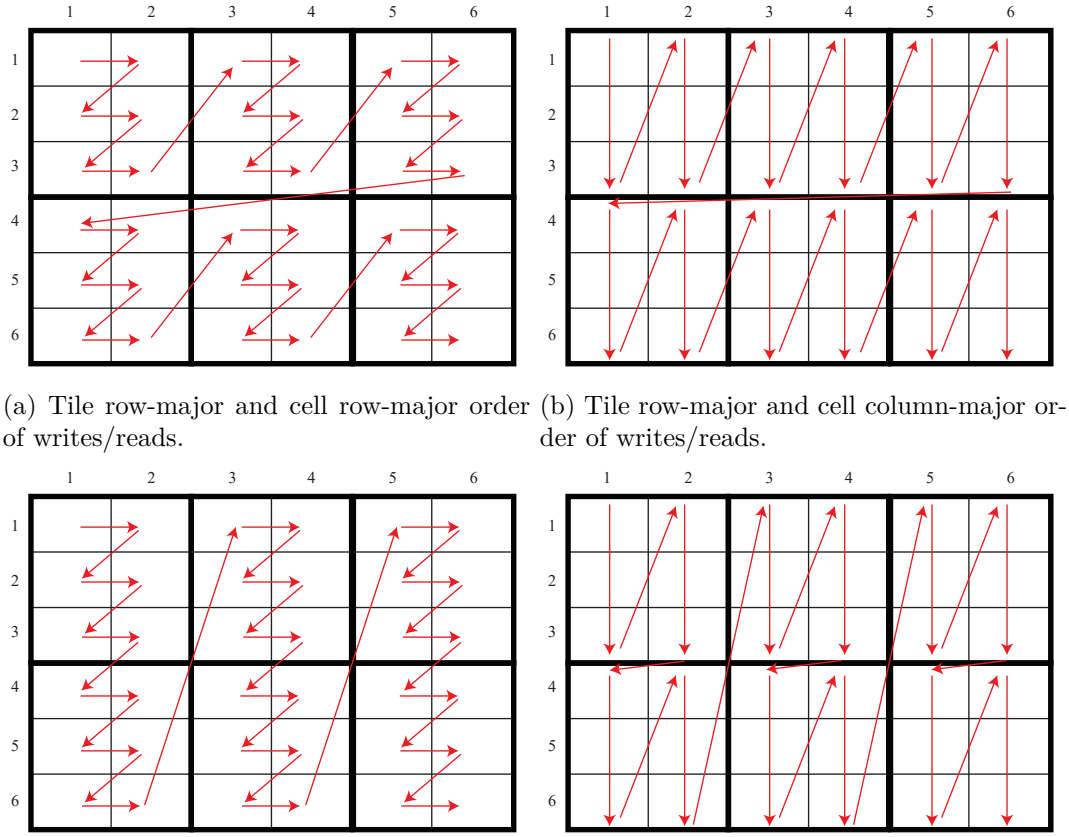


Figure 2.8: A tiled array.

Because of this, only four tiles of the array must be decompressed and retrieved (the tiles



(a) Tile row-major and cell row-major order of writes/reads.

(b) Tile row-major and cell column-major order of writes/reads.

(c) Tile column-major and cell row-major order of writes/reads.

(d) Tile column-major and cell column-major order of writes/reads.

Figure 2.9: The four possible combinations of tile/cell order.

that intersect the red rectangle). These logical tiles necessitate representation in the physical file as groups also of the size equal to the product of the tile extent that constitutes the space tiles. The arrangement of logical values in the physical file can follow two distinct orders:

- *Tile order*: the sequence in which tiles are read;
- *Cell order*: the sequence in which cells within each tile are read.

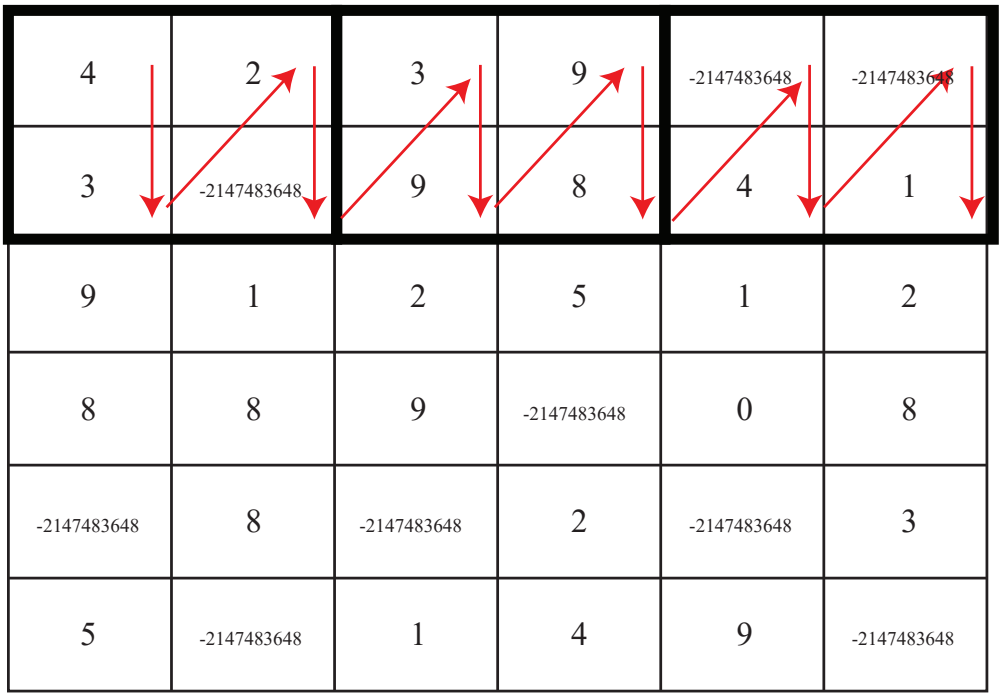
For both tile and cell ordering, there are two possible configurations: row-major (reading row by row) or column-major (reading column by column). Figure 2.9 illustrates the four potential combinations resulting from these orders.

Considering the array in Figure 2.6a, its physical representation with a tile extent of 2×2 will differ depending on both the tile/cell order. Sub Figure 2.10a of Figure 2.10 represents the physical representation of the first 12 values of this array in tile row-major/cell column-major order, and Sub Figure 2.10b shows the reasoning behind it.

Given TileDB's design to manage large datasets that may surpass main memory capacity, the consideration of tiles becomes essential. Section 2.5.7 will explain how the tile extent may affect performance.

4	3	2	-2147483648	3	9	9	8	-2147483648	4	-2147483648	1

(a) The physical representation of the array in tile row-major and cell column-major order.



(b) The reasoning for the ordering of the array.

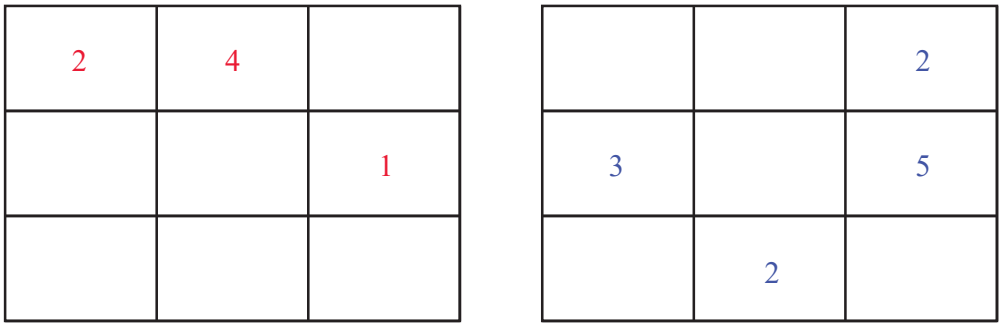
Figure 2.10: Representation of an array of tile extent 2×2.

2.5.3 Filters

Filters are mechanisms crafted to apply specific transformations on attributes prior to their recording in the array, such as compression[FIL].
The application of multiple filters is systematically orchestrated within a *filter list*, so that during the process of writing data to the array, the specified filters are applied in the order previously defined. Subsequently, the filtered data, when read from the array, undergoes a reverse unfiltering process, where the same list of filters is applied in the reverse order, resulting in the restoration of the original, unaltered data. The starting location of each filtered tile and the original tile size are both stored in the `__fragment_metadata.tdb` file.
Before initiating the filtering process for each data tile of an attribute, TileDB systematically partitions the tile into separate, non-overlapping chunks, which are subjected to independent filtering. This achieves cache optimization, as the chunk size can be strategically chosen to align with the L1 cache of processor cores, thereby enhancing its efficiency. This optimization proves especially beneficial for multi-stage filter lists, as the output from the preceding filter is more likely to remain in the L1 cache when utilized as input for the subsequent filter. Furthermore, this increases parallel computing capabilities, as unmistakable chunks can be filtered independently of one another.
The *compression* filter is independently applied to each attribute in TileDB, a critical consideration given the potential diversity in data types and values across attributes. TileDB provides a variety of compressors, such as GZIP, Zstandard, LZ4, RLE, Bzip2, and its proprietary implementation of double-delta compression. The selection of the most suitable compressor for a specific attribute presents, however, a significant challenge, as there is no universally ideal compressor; instead, the decision involves finding a balance between compression ratio and compression/decompression speed, a decision heavily influenced by the specific characteristics of the array data[COMa].

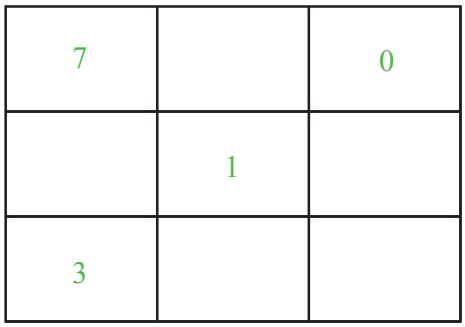
2.5.4 Encryption

TileDB offers a feature known as *at-rest encryption*, allowing the encryption of all attribute data and array metadata before persisting it. This encryption ensures that data and metadata are stored in an encrypted form on disk and are only decrypted in main memory when read[ENC].
When writing to an encrypted array, the process is similar to the unencrypted case. However, when the array is created or when it is opened for writing, the encryption algorithm and key must be specified. Reading from an encrypted array involves opening the array for reading with the correct encryption key, so that the decrypted (plaintext) result values are stored in memory. When compression or other filtering is configured, encryption is applied last.
TileDB never persists the encryption key, but it stores a copy of the encryption key in main memory while the encrypted array is open. When the array is closed, it zeroes out the memory used to store its copy of the key and frees the associated memory.
It currently supports AES-256 in the Galois/Counter mode (GCM) mode as the encryption algorithm, which is a symmetric, authenticated encryption algorithm.



(a) First write to the array.

(b) Second write to the array.



(c) Third write to the array.

Figure 2.11: A series of write operations to the same array.

2.5.5 Fragments and Consolidation

Inside the array directory, subdirectories named *fragments* are created after each write query. They hold the `.tdb` files, containing the written attribute information, along with associated metadata. The fragments pertain to the specific general name of `__<timestamp>_<timestamp>_<uuid>` , where `UUID` is a unique identifier for a process-thread pair and the timestamp records the time when the fragment was created[7].

Fragments are, thus, *immutable*, meaning a subsequent write operation never overwrites a file of a previously created fragment. Effectively, this means that fragments function as a versioning system for TileDB, each maintaining an array snapshot containing the cells written in the write operation.

In a read operation, TileDB chronologically imposes the most recent fragment (snapshot) as the one to be read, unless specified otherwise. Sub Figures 2.11a to 2.11c of Figure 2.11 represent the independent logical view of each fragment after a write operation.

Considering that each of these write operations were chronologically followed by one another from the first to the third write, the final logical view and the one that would be retrieved by default by TileDB can be observed in Figure 2.12.

The presence of numerous fragments can potentially impact TileDB's read performance (more on this in Section 2.5.7). To address this issue, TileDB offers *consolidation*, a feature that

[7]The reasoning for two timestamps will be explained below, but after a write operation, both timestamps will match.

7	4	0
3	1	5
3	2	

Figure 2.12: The logical view of the array after the three writes.

allows the fusion of existing fragments into a single one. Note that, now, the timestamps of the fragment's filename do not match anymore, since the first corresponds to the first timestamp of the first fragment that was consolidated and the second to the first timestamp of the last fragment that was consolidated.

2.5.6 Parallelism and Concurrency

TileDB not only incorporates extensive internal parallelization to optimize the queries' performance, ensuring scalability for hardware with parallelism capabilities, but also multiprocessing capabilities for user programming.
The following parallelization capabilities are ensured[PAR]:

- Read and write parallelism: TileDB parallelizes both read and write queries to efficiently utilize available cores;
- Filtering parallelism: During read queries involving filtering, such as compression, TileDB adopts a nested parallelism strategy. The strategy is specified in Algorithm 1.

Algorithm 1 TileDB's parallelism algorithm for filtering.

```
Let A be the list of all attributes pertaining to the array.
Parallelize for each α in A do
    T ← getAttributeTiles(α)
    Parallelize for each t in T do
        C ← divideTileIntoChunks(t)        ▷ 64KB chunks, the common size of L1 caches.
        Parallelize for each c in C do
            Filter(c)
        end for
    end for
end for
```

As for concurrency, there may be three cases for concurrency, as listed in Table 2.6.

Scenario	Thread-/Process-safe?	Explanation
Array Creation	No	TileDB considers multiple threads or processes attempting to define the same array in parallel as considered impractical, especially considering it is an exceptionally lightweight process, whose performance is not anticipated to construct a significant bottleneck.
Writing	Yes	All write operations are atomic, regardless of array type, as it involves writing data to an array slice, thus, concurrent writes are possible due to each thread/process creating a separate fragment upon the first submission of a write query (invisible until the query is concluded), whose name is unique by incorporating the unique UUID per process/thread and the current timestamp; this eliminates any need for synchronization or locking, but still avoids all possible conflicts, even at the file system level.
Reading	Yes	Similar to the writing operations, all read operations are atomic too; in multiprocessing settings, concurrent reads are independent and require no locking, however, in multi-threading scenarios, locking (using mutexes) is employed only when queries access the tile cache, resulting in minimal overhead. Note that, because of this, concurrent reads and writes can be freely interleaved.
Consolidation	Yes	Consolidation can also be performed concurrently with other reads and writes, as it occurs independently of them, as the new fragment is not visible until consolidation is complete. Locking is only enforced by a brief period after consolidation, when old fragments are deleted, and the new fragment becomes visible. TileDB employs, again, mutexes for locking in multi-threading, but file locking for multiprocessing.

Table 2.6: TileDB's concurrency scenarios.

2.5.7 Performance Considerations

When designing the array schema in TileDB, it is crucial to account for various factors that can impact performance, as its effectiveness is inherently tied to the essence of the data the array will accommodate and the specific types of queries expected to be performed. Crafting a well-suited schema for a TileDB array involves considering a series of factors, such as:

- *Sparse arrays vs. Dense arrays*: sparse arrays incur additional overhead due to the necessity of explicitly storing the coordinates of non-empty cells. Depending on the use case, this additional overhead may be justified by the storage efficiency gained from not storing dummy values for empty cells, as required in the dense alternative. It is imperative to consider the nature of the data in the array. If most cells in the array have non-empty values, dense arrays may be more suitable, but if the opposite is true, sparse arrays become the preferred choice for most cases;
- *Space tile extent*: in both dense and sparse arrays, the space tile shape and size play a crucial role in determining their atomic unit of I/O and compression. In dense arrays, all space tiles that intersect a read query's slice are thoroughly read from disk, even if they only partially intersect with it, therefore, optimizing the space tile shape and size to closely match the typical read query can significantly enhance performance.

Similarly, in sparse arrays, the space tiles have a significant impact on the order of the non-empty cells and the shape and size of the minimum bounding rectangles (MBRs), as all tiles in a sparse array whose MBRs intersect a read query's slice are read from disk, hence, selecting space tiles in a way that shapes the resulting MBRs to be similar to the read access patterns can lead to improved read performance as well;

- *Tile Capacity*: determines the number of cells in a data tile of a sparse fragment, serving as the atomic unit for I/O and compression. If the tile capacity is too small, it may result in the generation of numerous small and independent read operations to fulfill a specific query; additionally, very small tiles can lead to poor compressibility, impacting the overall storage size. On the other hand, if the tile capacity is too large, I/O operations become less frequent but larger in scale. While larger tiles are generally favorable, excessive capacity may lead to wasted work, as more data is read than necessary to fulfill a typical read query;

- *Dimensions and Attributes*: when read queries consistently access the entire domain of a particular dimension, a more effective design choice may be to consider making that dimension an attribute, as dimensions are more suitable when read queries involve retrieving specific ranges of cells based on that dimension, which TileDB optimizes for. Another point to consider is that subselecting on dimensions is not supported, as, while a read query can specify a subset of attributes to be read, it must always specify all dimensions;

- *Filtering*: filtering, including compression, applied to attributes can lead to a reduction in persistent storage consumption, at the cost of increased time required for reading and writing tiles to and from the array (compression/decompression time). Despite the potential overhead in time, the smaller storage may be preferred, especially since smaller tiles on disk may be read quicker, ultimately enhancing performance. Furthermore, there exist several compressors, with different levels of compression, so it is important to fine-tune for the specific data being stored, reaching a balance between storage space and speed;

- *Number of Array Fragments*: Each write operation in TileDB results in the creation of a new fragment. During read queries, TileDB considers all fragments and sorts them by timestamp to ensure that the latest data is returned, so, when dealing with numerous fragments, a performance overhead may occur, as, internally, TileDB must conduct sorting operations and determine potential overlaps of cell data across all fragments during read queries. The user must decide between keeping a versioning system using the fragments, or consolidate them if performance is a bigger problem, leading to improved read performance;

- *Read/Write layouts*: if an ordered layout is specified for a read/write query and that layout differs from the array's physical layout, TileDB must internally sort and reorganize the cell data that is read/written from/to the array into the requested order.

TileDB may skip this process if the global order query layout is specified instead, saving time;

- *Opening and closing arrays*: an array must be opened before any queries can be issued to it; after being opened, TileDB reads and decodes the array and fragment data, which may involve disk operations and potentially decompression, so, it may be worthwhile to issue all the needed queries without closing the file, so that that overhead may not occur.

2.6 Conclusion

In conclusion, TileDB has been identified as a possible contender to RINEX to save the GNSS data gathered by the satellites. This data must be stored efficiently in order to achieve an easy data management, even if the amount of data is significant.

Chapter 3

State-of-the-Art

3.1 Introduction

This chapter seeks to explore the contemporary research of TileDB, as a GNSS storage framework. Section 3.2 explains why TileDB is being considered for storing GNSS data, outlining other possible storage methods and why TileDB is superior. Section 3.3 explains UNAVCO's approach to using TileDB in a cloud-based model to store and exchange GNSS data. Finally, Section 3.4 presents a software developed by GFZ to upload RINEX data into TileDB.

3.2 Why Consider TileDB for Storing GNSS Data?

TileDB has several useful features that make it well-suited for storing GNSS data, such as:

- Reduces storage needs through sparse arrays: GNSS data is, by definition, sparse (see Figure 3.1), as in any instance, not all possible satellites may be available, causing "gaps" in the data; by using sparse arrays, it is ensured that those empty values do not occupy unneeded storage;
- Supports fast multidimensional slicing (via tiling): read operations are simplified by slicing between the dimensions needed, and TileDB incorporates efficient algorithms that significantly accelerate read operations compared to the outdated method of reading from files;
- Supports efficient multi-threading: which further hastens read/write operations;
- Supports versioning through fragmentation: old data is preserved through the use of fragments, allowing it to be queried by specifying the fragments' timestamp, which may be useful, as even previously incorrect data may be required in an archive, even if it has been corrected;
- Leading innovators in the GNSS data storage research area, such as UNAVCO (see Section 3.3), have considered and implemented early prototypes using TileDB.

3.2.0.1 TileDB vs. HDF5

The hierarchical data format version 5 (HDF5) is a data model, library, and file format designed for storing and managing diverse data types efficiently. It introduces groups and datasets, which provide flexibility for high-volume and complex data[FHK+11]. Figure 3.2 shows an overview of HDF5's data model.

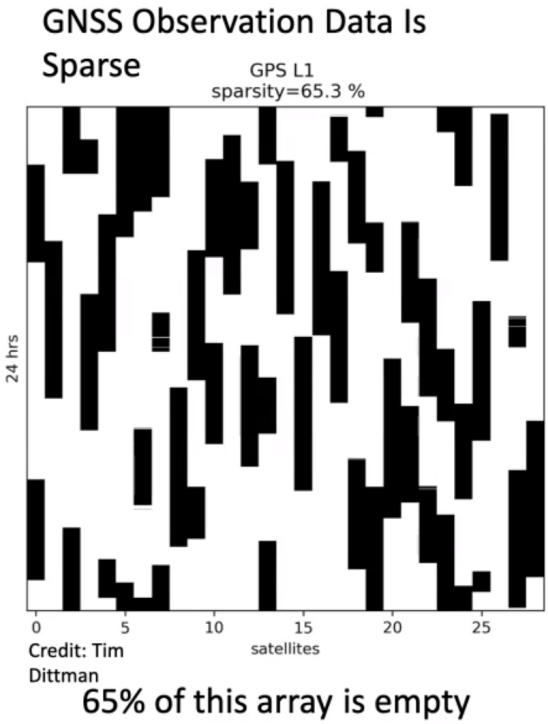


Figure 3.1: An example of "sparse" GNSS data.

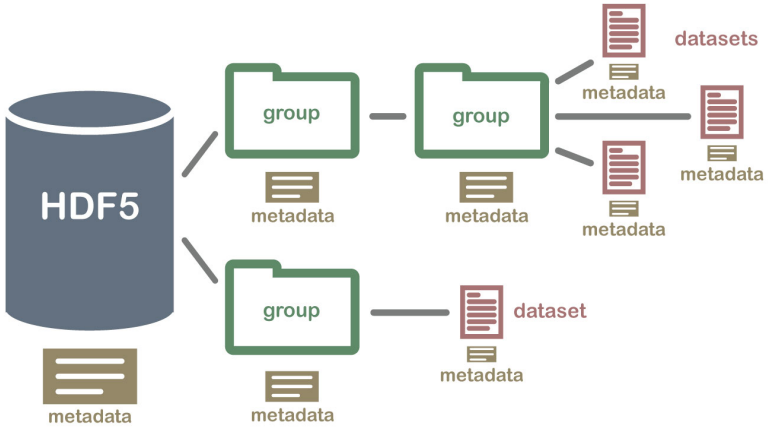


Figure 3.2: HDF5's data model overview. From Neonscience in "Hierarchical Data Formats - What is HDF5?", 2020[Was].

A study by Yiyu Ni et al.[NDF+23] found improved performance in compute time and memory throughout when using TileDB compared to the conventional HDF5 data format. According to Stavros Papadopoulos et al.[PDMM16], HDF5 has two main shortcomings that TileDB solves: first, it struggles to efficiently capture sparse arrays, as it does not support them natively as TileDB does; secondly, HDF5 is optimized for in-place writes of large blocks, which results in poor performance when writing small blocks of elements that are randomly distributed, primarily due to the expensive random disk accesses they incur.

3.2.0.2 TileDB vs. Zarr

Closely aligned with the HDF5 model, Zarr adopts a similar hierarchical structure, organizing groups of datasets with associated metadata at each level. A distinctive feature of Zarr, in contrast to HDF5, lies in its utilization of individual files for each dataset, employing a simplified internal binary data structure[MK23].
There are no meaningful studies comparing Zarr to TileDB yet, leaving a notable gap in the understanding of their relative strengths and weaknesses. Further research in this area is essential for providing comprehensive insights into the optimal use cases of Zarr on storing GNSS data.

3.3 UNAVCO's Approach

The University NAVSTAR Consortium (UNAVCO) is a non-profit university-governed consortium, which facilitates geoscience research and education using geodesy[UNAa]. It belongs to EarthScope Consortium, a community of scientists, educators, and professionals working together to understand Earth processes and hazards using geophysics[UNAc]. The volume of GNSS data is experiencing exponential growth, nearing 1 petabyte of data and consisting of approximately 35 million discrete RINEX files for UNAVCO.
UNAVCO intends to address RINEX's problems by creating a system that encompasses[Ber23]:

- An *analysis-ready and cloud-optimized* (ARCO) storage system: the primary objective is not to eliminate RINEX, as several processing tools and communities heavily rely on them for their operations; instead, they aim at establishing an improved archiving method that is cloud-ready;
- Separating metadata storage from GNSS data: they aim at storing metadata independently of the data, although concerns about synchronization and consistency are acknowledged;
- Creating a real-time cloud system: the organization is working towards the development of a real-time cloud system to enhance the management and accessibility of GNSS data by third-party communities.

Currently, they have been in development for a year with a team of 20, working on the system depicted in Figure 3.3.
Stations are streaming real-time data (0), and they are periodically downloading the data from stations as well (1). That data goes to a scheduled task in amazon web services (AWS) Fargate, where it uploads all the data to an AWS simple storage service (S3) bucket (3). Through AWS simple notification service (SNS), they send a notification to some message queues in AWS simple queue service (SQS) with some metadata of the RAW files uploaded to the bucket (4) and then, through the converters, they are converted to RINEX and then converted to TileDB arrays (through lambda functions) also stored on AWS S3 buckets. Then, an application will mark the file as downloaded, ensuring it does not get repeatedly downloaded (5). They do this in batches to avoid overloading the system.

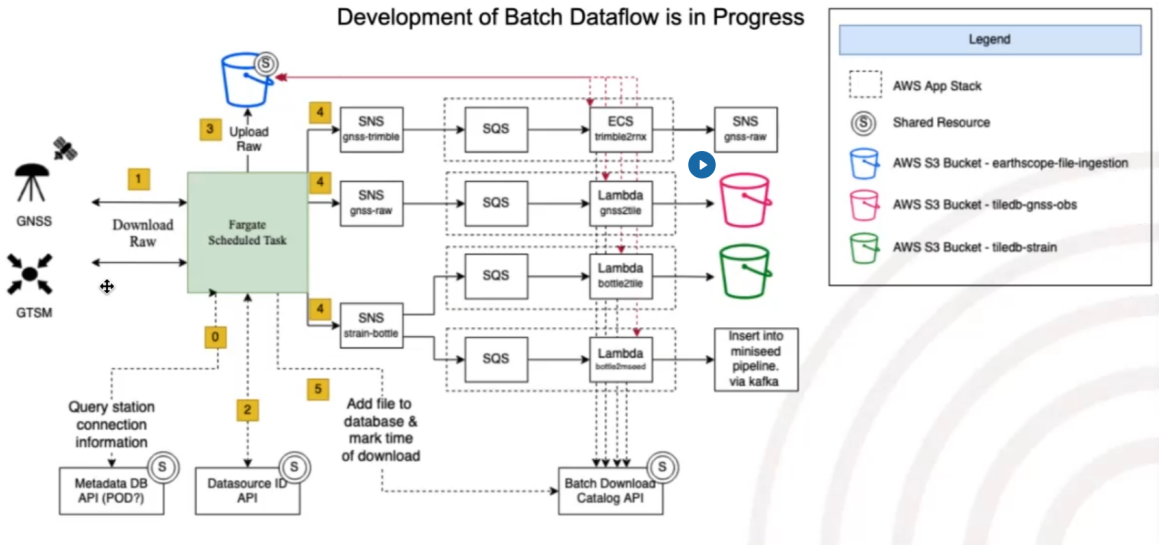


Figure 3.3: UNAVCO's TileDB ARCO real-time storage system prototype using AWS. From Henry Berglund in "TileDB"[Ber23].

3.3.0.1 UNAVCO's TileDB data model

The UNAVCO's TileDB prototype comprises four dimensions[UNAb]:

- *Time* (POSIX);
- *System* (GPS, GLONASS, GALILEO, etc.);
- *Satellite number*;
- *Observation codes*: type (C,L,D,S,X), Band/Freq (1,2,...,9) and Attribute (W, I, Q, C, P, ...).

It is easy to slice for time, system, satellite, and observation codes with this design. For each set of coordinates in these four dimensions, seven attributes are defined:

- *Pseudorange* (meters,float64);
- *Phase* (cycles, float64);
- *Doppler* (Hz, float64);
- *Signal-to-noise* (dbHz, float32);
- *Slip* (count, uint16);
- *Flags* (uint16): LLI, multi-path mitigation, smoothing, etc;
- *frequency channel number* (FCN) (byte).

The design aims to have an array per station, as too many dimensions may cause performance issues. To query all observations from a specific time period while disregarding the stations, an external catalog is maintained and used for multi-query purposes.

3.4 GFZ's Pynex

The *GeoForschungsZentrum* (GFZ) is the national research center for Earth sciences in Germany[GFZ]. They have been developing an application that transforms a RINEX file into a Python Pandas DataFrame, which can subsequently be imported into TileDB. The application is designed to convert RINEX observation, navigation, and meteorological files into DataFrames and supports the parsing of both RINEX 3/4 observation/meteorological files and RINEX 4 navigation files. It can be used from the command line by issuing the command `pynex read RINEX_FILE(S)`[Brab].

3.5 GFZ's GNSS TileDB

GFZ has also developed an application that creates a TileDB schema and enables bidirectional conversion between RINEX and TileDB formats by using previously created DataFrames (possibly using Pynex)[Braa].

3.6 Conclusion

In summary, many investigators have started prototypes for TileDB as a novel methodology for storing GNSS data, with possible increases in efficiency over RINEX. It is clear that they are still in early levels of development and its advantages are not yet immediate.

Chapter 4

Raw2Rin Software Package

4.1 Introduction

To address the challenge posed by numerous proprietary formats and their respective conversion tools, the software package `raw2rin` was developed. It is composed of three different applications and aims at creating an easy-to-use framework, where users are able to:

- Convert RAW files from several proprietary formats to RINEX;
- Change conversion parameters;
- Correct the RINEX's metadata;
- Manage different jobs.

This chapter provides a detailed look at its architecture, explaining how its components integrate to provide functionality to users.
Section 4.2 introduces the overall system architecture, illustrating how the three main applications of the software package, the client interfaces, web API, and converter application, are interconnected. Sections 4.3, 4.4, and 4.5 describe the specific roles, operations, and design of each component: the client applications for user interaction, the web API for request handling and data management, and the converter application for data processing and storage.
Finally, Section 4.6 provides a summary of the chapter, recapping the primary functions and collaborative flow of each component in the ecosystem.

4.2 Application Architecture

The architecture of the developed application is shown in Figure 4.1.
It is designed as a cloud-based system, enabling clients–whether web- or command-line-based–to submit requests to a cloud-hosted web server via a RESTful API. The Web server subsequently communicates with a cloud-hosted SQL database to manage data associated with user-created jobs, facilitating both data entry and retrieval. Additionally, a converter application hosted on a separate cloud server processes client requests, allowing users to access results by interacting with the Web server.

4.3 Client Applications

Users may interact with the application through two primary interfaces: a command-line application and a Web client. Although the command-line application integrates some TileDB

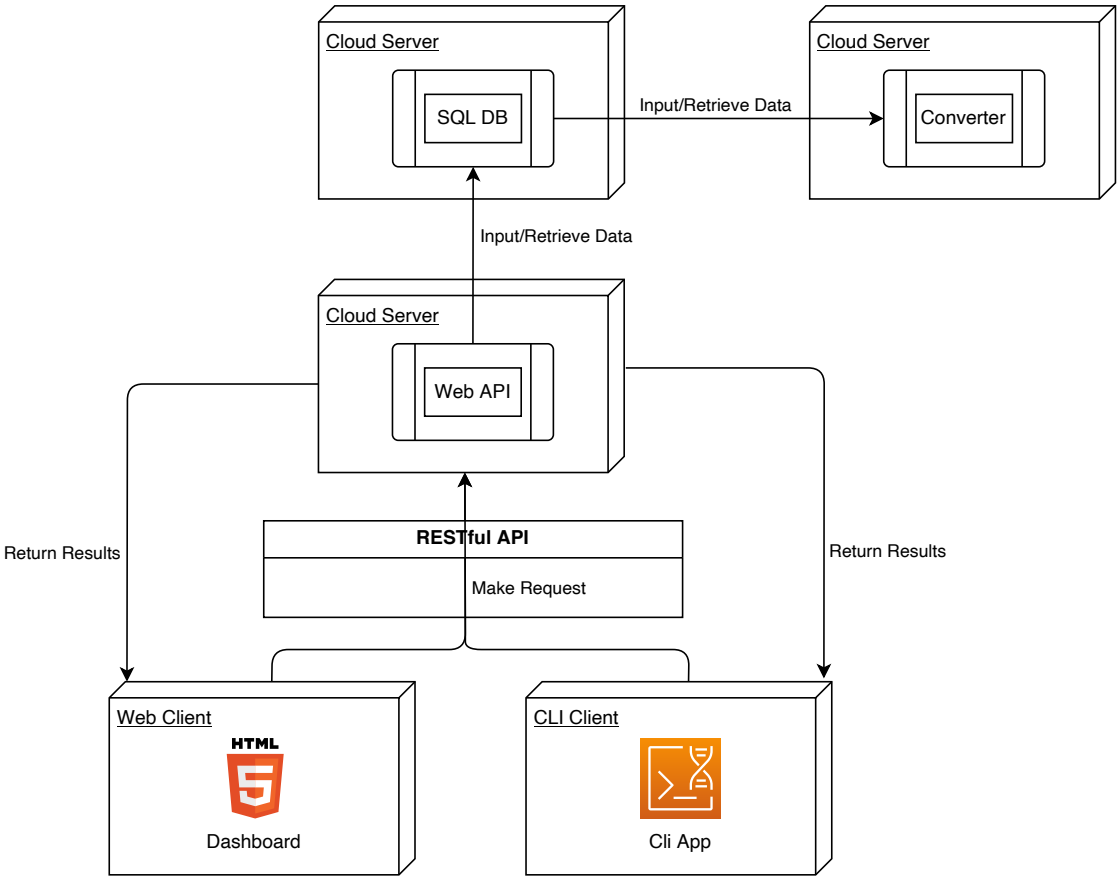


Figure 4.1: The application's system architecture.

functionality explained in Section 5, all other requests are available in both client types and function identically. The Web client is optimized for users seeking quick conversions or for those with limited experience, while the command-line interface caters to advanced users, particularly those processing large volumes of GNSS data or automating tasks.

Regardless of the chosen interface, users initiate API requests by specifying required parameters, each corresponding to various actions, including user management, data processing tasks, etc. In the command-line application, users may employ the `-r` flag to designate the type of request, followed by additional flags that represent the parameters required for that request.

Each API interaction returns a default raw JavaScript object notation (JSON) response that includes either a message field, signaling a successful operation, or an error field, which provides structured feedback in the case of a failure. When the request is successful, the message field provides a confirmation of the completed action. In contrast, if the request encounters issues, the error field contains a dictionary, where each key represents an input field and includes an array of error messages describing the validation issues. For instance, a registration attempt with duplicate information or invalid input (such as a username that is already in use, an email conflict, or a password with less than the required minimum of eight characters) will trigger an error message, as shown in Listing 4.1:

Listing 4.1: Returned API errors in JSON format.

```
{
  "error": {
    "email": ["The email 'xpto@example.com' is already taken. Please choose a different one
        ↪ ."],
    "username": ["The username 'xpto' is already taken. Please choose a different one."],
    "password": ["Field must be between 8 and 255 characters long."]
  }
}
```

The command-line client application supports file-based requests, offering two methods to specify input files. By default, file paths are expected to be enclosed in double quotes, enabling pattern matching with wildcard characters[1]. Alternatively, users can list multiple file paths directly in a comma-separated list by using the `-np` flag.
When the `-pr` or `--pretty` flag is included in a request, the output is formatted for readability and displayed directly in the terminal, providing a structured and user-friendly representation of the response data. In the absence of this flag, the default raw JSON response string is printed instead.
Data input in the Web application is managed through forms, with each field corresponding to a specific request parameter. Upon submission, success and error messages are displayed directly in the browser as flash pop-up windows.
Uploaded files, along with the generated results, are managed by an automated retention policy, where files are removed from the server under ond of the following conditions: 1) one full day (24 hours) has passed since file upload, 2) the user downloads the generated results, 3) the processing job encounters an error, or 4) the user explicitly issues a `delete` or `clean` request.
The following sections offer a comprehensive breakdown of each available request type, including the necessary input parameters, anticipated response formats, and practical usage examples.

4.3.1 Register

A `register` request is used to create a new user account within the system. This request requires the submission of personal details, including the user's name, username, email, and password.

4.3.1.1 Command-Line Application

Listing 4.2 shows the general syntax for the `register` request within the command-line application.

Listing 4.2: General syntax of the register request.

```
./raw2rin -r register -n NAME -u USERNAME -e EMAIL -p PASSWORD [-pr]
```

The flags are:

- `-n`: Specifies the full name of the user to register;

[1]This behavior mirrors Unix-based commands, such as `ls`.

- `-u`: Specifies the username for the new user account;
- `-e`: Specifies the email address for the account;
- `-p`: Specified the password for the account.

4.3.1.2 Web Application

Figure 4.2 shows the register page on the Web client. For a full-sized image, check Appendix A.1.

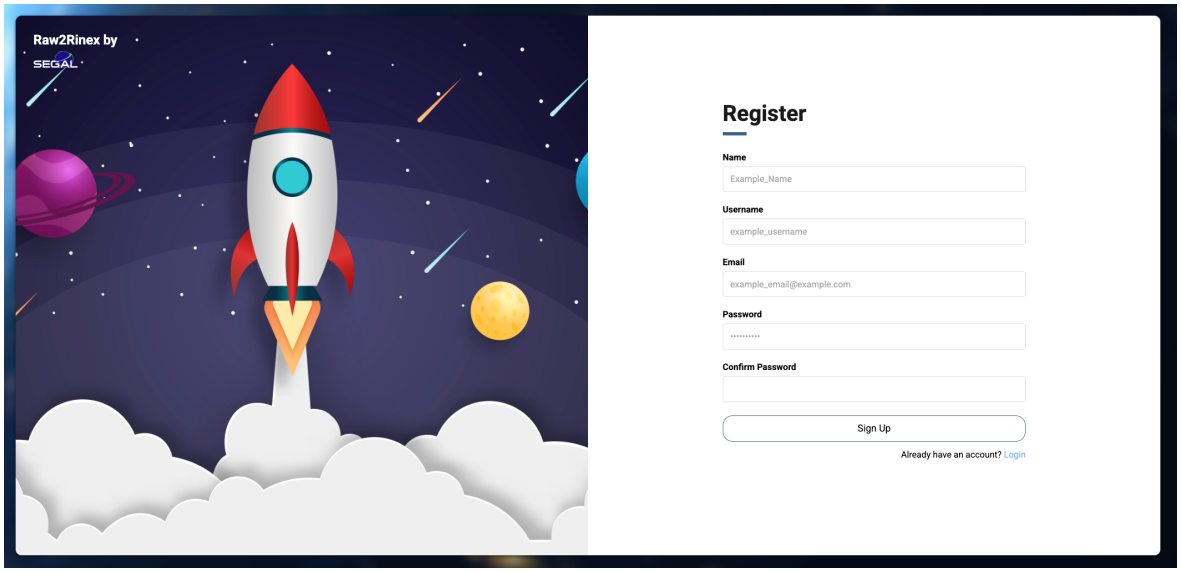


Figure 4.2: The Web client register page.

It contains the form where users input the required parameters for the `register` request. The parameters are identical to those in the command-line application, with the addition of a `Confirm Password` field to ensure password accuracy and prevent typographical errors.

4.3.2 Login

A `login` request authenticates the user in the system. In the command-line application, it generates a token for the user, enabling access to requests that require authorization, whereas the web-based clients gain access to authentication cookies. Please check section 4.4.3 to learn more.

4.3.2.1 Command-Line Application

Listing 4.3 shows the general syntax for the `login` request within the command-line application.

Listing 4.3: Login command structure

```
./raw2rin -r login -ue USERNAME_EMAIL -p PASSWORD [-pr]
```

The flags are:

- `-ue`: Either the user's username or email;
- `-p`: The password of the user.

Upon successful authentication, the response returns a JSON message containing the authentication token alongside a confirmation message. For error responses, only the standard error message structure mentioned in section 4.3 is returned.
Listing 4.4 shows an example of a successful login response.

Listing 4.4: Example of a successful login response

```
{
    "message": "You have been successfully authenticated. Please use the following token in
        ↪  your future requests. If it doesn't work, please reauthenticate, as the token
        ↪ may have expired.",
    "token": "7a81e2953bcc868ba05532df793f241e"
}
```

4.3.2.2 Web Application

Figure 4.3 shows the login page on the Web client. For a full-sized image, check Appendix A.1.

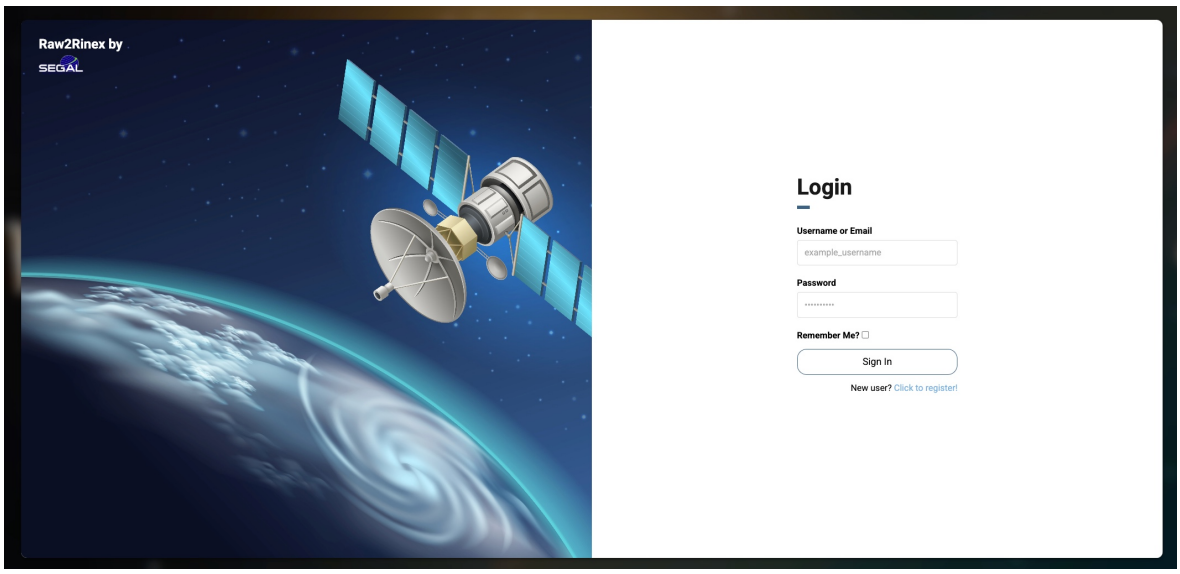


Figure 4.3: The Web client login page.

It contains the form where users input the necessary parameters for the `login` request. The parameters are identical to those in the command-line application, with the addition of a `Remember Me?` checkbox, allowing users to remain consistently logged in across sessions.

4.3.3 Convert

A `convert` request is used to convert one or more RAW files into a single RINEX file.

4.3.3.1 Command-Line Application

Listing 4.5 shows the general syntax for the `convert` request within the command-line application.

Listing 4.5: Convert command structure

```
./raw2rin -r convert -pf PROPRIETARY_FORMAT -v VERSION -i INPUT_FILES…, -t TOKEN [-np] [-pr
    ↪ ]
```

It contains the following flags:

- `-pf`: The proprietary format in which the RAW input file(s) were encoded. The current implemented options are: `sbf`, `t02`, `m00`, and `cnb`;
- `-v`: The version of the resulting RINEX file. Note that not all versions are supported by all proprietary formats, so an error may occur if an unsupported version is assigned. Only single-numbered versions are supported, and they will translate to either the default or latest version in the respective converter;
- `-i`: The input RAW files. Each file will be converted separately and then joined together, resulting in a single RINEX file;
- `-t`: The token used for user authentication.

Listing 4.6 shows an example of a `convert` request command and Listing 4.7 shows an example of a `convert` request command with multiple files.

Listing 4.6: Example of a Convert Request command

```
./raw2rin -r convert -pf sbf -v 3 -i "BUMT202012191700B.sbf.gz" -t
    ↪ f3518eef004568f34eb973466504ab2c
```

Listing 4.7: Example of a Convert Request command with multiple files

```
./raw2rin -r convert -pf sbf -v 3 -i BUMT202012191700B.sbf.gz,BUMT202102010100A.sbf -np -t
    ↪ f3518eef004568f34eb973466504ab2c
```

Upon successful processing of the request, the response returns a JSON message containing a job identification number (ID) (more on this in section 4.4.2), which can be used to check the status of the job and download its results after processing. For error responses, only the standard error message structure mentioned in section 4.3 is returned.
Listing 4.8 shows an example of a `convert` success response.

Listing 4.8: Example of a successful convert response

```
{
    "job_id": 77810,
    "message": "Your files have been queued for upload!"
}
```

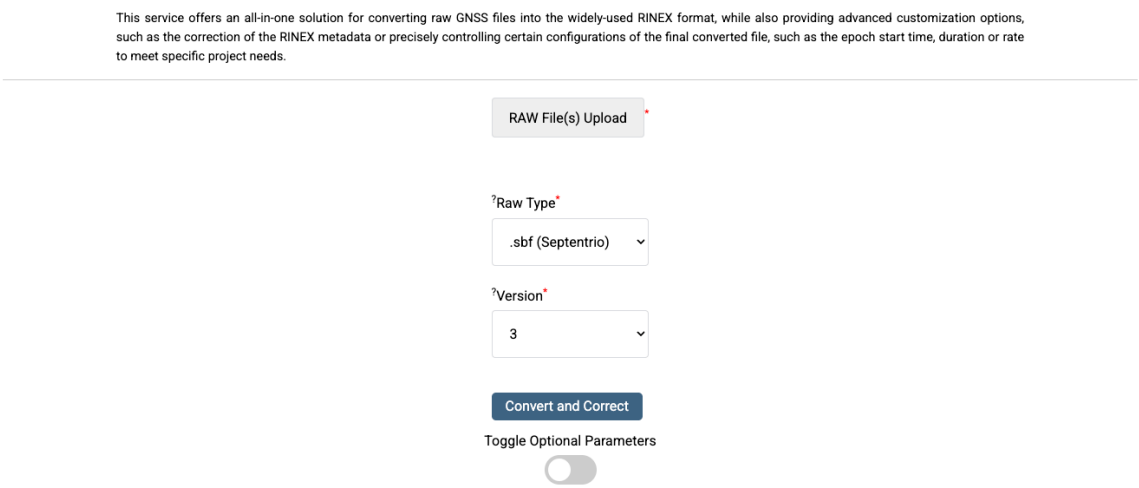


Figure 4.4: The Web client convert page.

4.3.3.2 Web Application

Figure 4.4 shows the general convert page of the Web client. For a full-sized image, check Appendix A.1.
Since the optional parameters slide is turned off, the system will exclusively execute the conversion process without incorporating any additional modifications.
All mandatory fields are clearly marked, and some include a question mark icon that provides concise descriptions of the respective parameters. The RAW files can be uploaded using the multiple file select field. The two other parameters are implemented as select dropdowns, offering users a predefined list of possible data types and corresponding version options.

4.3.4 Correct

A `correct` request is used to modify some parameters of a RINEX file and optionally change its metadata according to user-specified data.

4.3.4.1 Command-Line Application

Listing 4.9 shows the general syntax for the `correct` request within the command-line application.

Listing 4.9: Correct command structure

```
./raw2rin -r correct -i INPUT_FILE -t TOKEN [-rt RATE] [-es EPOCH_START] [-d DURATION] [-
    ↪ iso3 COUNTRY-CODE] [-s SETTINGS_FILE] [-pr]
```

It contains the following flags:

- `-i`: A single uncompressed RINEX file;
- `-t`: The token used for user authentication;
- `-rt`: The new rate value of the resulting RINEX file;

- `-es`: The new epoch start date value of the resulting RINEX file;
- `-d`: The new duration time value of the resulting RINEX file;
- `-iso3`: The new country code for the resulting RINEX file;
- `-s`: A JSON file containing new values for the metadata. If certain metadata parameters are not provided, they will remain unchanged.

Listing 4.10 shows an example of a `correct` request command.

Listing 4.10: Example of a Correct Request command

```
./raw2rin -r correct -rt 60 -es 2020-01-01_14:11:12 -d 3600 -iso3 BTN -s settings.json -i
    ↪ BUMT00BTN_R_20203541700_01H_30S_MO.rnx -t f3518eef004568f34eb973466504ab2c
```

A boilerplate for the settings file is displayed in Listing 4.11.

Listing 4.11: Boilerplate for the settings file.

```
{
  "marker": "new_marker_name",
  "longMarker": "new_longMarker_name",
  "X": "new_X_name",
  "Y": "new_Y_name",
  "Z": "new_Z_name",
  "dE": "new_dE_name",
  "dN": "new_DN_name",
  "dU": "new_dU_name",
  "markerNumber": "new_markerNumber_name",
  "agency": "new_agency_name",
  "siteOperator": "new_siteOperator_name",
  "runBy": "new_runBy_name",
  "antType": "new_antType_name",
  "antSN": "new_antSN_name",
  "recType": "new_recType_name",
  "recSN": "new_recSN_name",
  "recFW": "new_recFW_name"
}
```

The response is the same as the `convert`'s.

4.3.4.2 Web Application

Figure 4.5 shows the general correct page of the Web client. For a full-sized image, check Appendix A.1.
All mandatory fields are clearly marked, and some include a question mark icon that provides concise descriptions of the respective parameters. The form allows the upload of a single RINEX file through a dedicated file select field. Additionally, a settings file can be optionally uploaded using another single file select field. To simplify customization, a button is provided to download a boilerplate version of the settings file for reference or editing.
All remaining parameters are optional and pre-configured with default values; if left unchanged, these defaults ensure the system behaves as though the parameters were not selected. The rate field is implemented as a dropdown menu featuring commonly used rate values, the epoch start and duration fields enable users to define a specific epoch period for

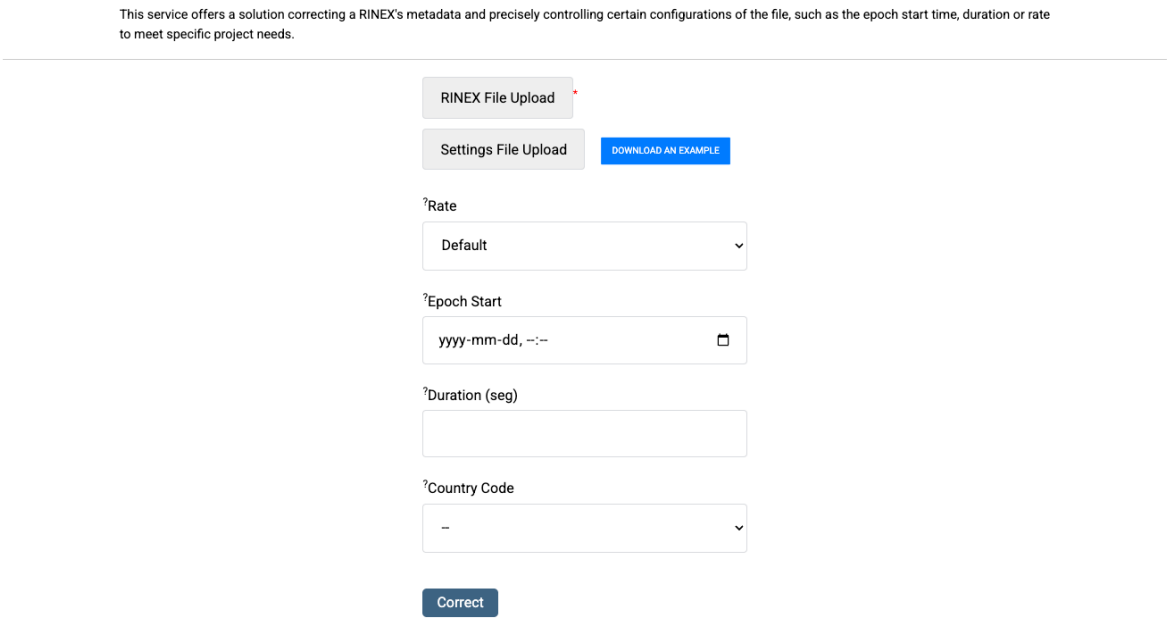


Figure 4.5: The Web client correct page.

the resulting RINEX file, and the country code field includes a comprehensive list of all ISO 3 country names.

4.3.5 Convert and Correct

The `convert_correct` request merge the `convert` request with the `correct` request. Hence, it converts one or more RAW files into RINEX, modify its parameters, and optionally change its metadata parameters.

4.3.5.1 Command-Line Application

Listing 4.12 shows the general syntax for the `convert_correct` request within the command-line application.

Listing 4.12: Convert and Correct command structure

```
./raw2rin -r convert_correct -pf PROPRIETARY_FORMAT -v VERSION -i INPUT_FILES…, -t TOKEN [-
    ↪ rt RATE] [-es EPOCH_START] [-d DURATION] [-iso3 COUNTRY-CODE] [-s SETTINGS_FILE] [-
    ↪ np] [-pr]
```

It contains the same flags previously explained in Sections 4.5.2 and 4.5.3.
Listing 4.13 shows an example of a `convert_correct` request command.

Listing 4.13: Example of a Convert and Correct Request command

```
./raw2rin -r convert_correct -rt 60 -es 2020-01-01_14:11:12 -d 3600 -iso3 BTN -s settings.
    ↪ json -pf sbf -v 3 -i "data/MAPX053*" -t f3518eef004568f34eb973466504ab2c
```

Similarly to the `convert` and `correct` requests, this request has the same success and error messages previously detailed in section 4.3.

4.3.5.2 Web Application

Figure 4.6 shows the general convert and correct page of the Web client. For a full-sized image, check Appendix A.1.

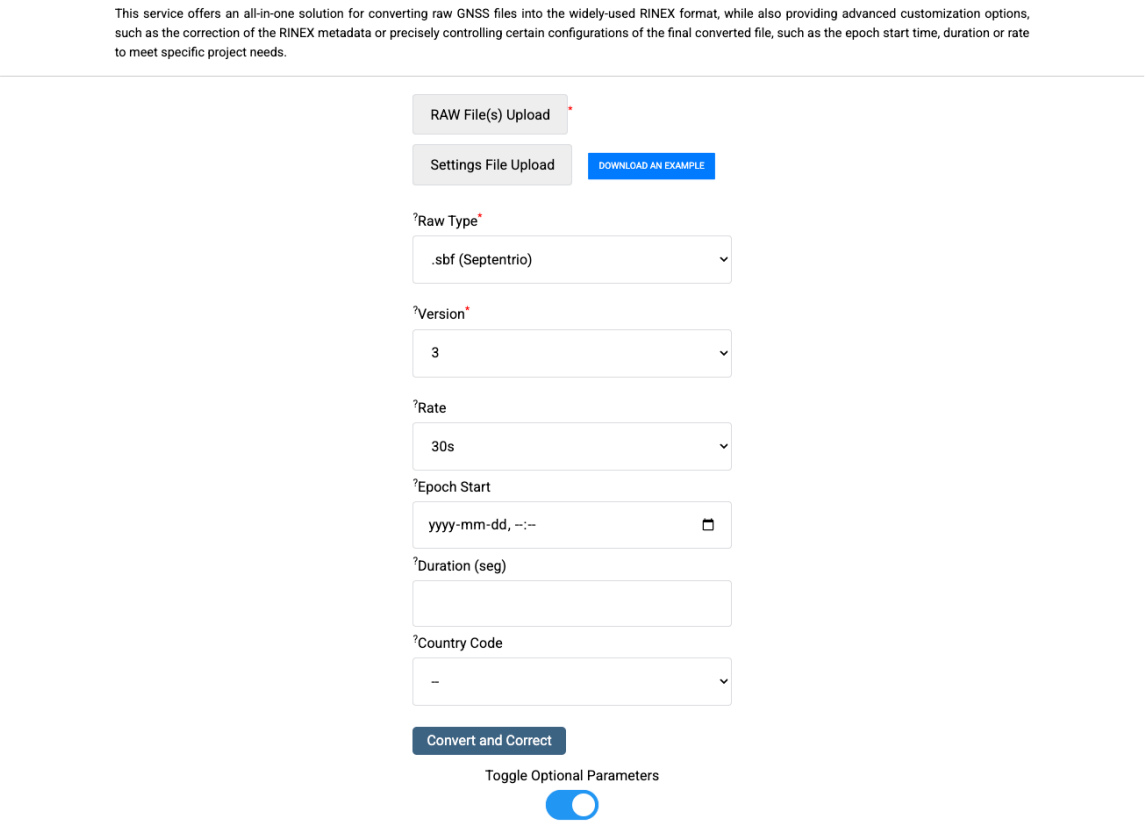


Figure 4.6: The Web client convert and correct page.

With the optional parameters slide enabled, the system will execute the conversion process and apply any additional changes specified by the user.
The fields in this configuration mirror those outlined in Sections 4.3.3.2 and 4.3.4.2. The key distinction is that the rate parameter becomes mandatory when multiple files are uploaded, ensuring consistent processing across all selected files.

4.3.6 Job

A `job` request retrieves the details of a single job that the user has submitted. This allows users to monitor a specific job's progress.

4.3.6.1 Command-Line Application

Listing 4.14 shows the general syntax for the `job` request within the command-line application.

Listing 4.14: Job Request command structure

```
./raw2rin -r job -jid JOB_ID -t TOKEN
```

The flags are:

- `-jid`: The job's ID of the job to retrieve, obtained from any processing request;
- `-t`: The token used for user authentication.

Besides the default JSON string previously defined in section 4.3, there does not exist any more output for error responses. In a successful response, however, the JSON output includes a `job` field, containing the following fields:

- `job_id`: The unique identifier for the job;
- `job_type`: The type of job;
- `status`: The current status of the job (more on this in section 4.4.2);
- `err_msg`: The error message in case the status of the job is `error`.

For instance, Listing 4.16 shows the successful response for the `job` request of listing 4.15.

Listing 4.15: Example of a Job Request command

```
./raw2rin -r job -jid 33 -t f3518eef004568f34eb973466504ab2c
```

Listing 4.16: Example of a successful response for Job Request

```
{
    "job":{
        "job_id":33,
        "job_type":"convert",
        "status":"error",
        "err_msg":"The version '1' is not valid. Only versions '2, 3, 4' are allowed for
            ↪ RAW files sbf. Please check the input version and try again".
    },
    "message":"Your job has been successfully retrieved!"
}
```

4.3.6.2 Web Application

The request does not have a dedicated web client page because it is designed to provide a more streamlined and efficient user experience.
By consolidating all job-related details in one place, users can quickly review and manage their tasks without switching between different pages, which minimizes potential confusion and improves usability.
Furthermore, it helps optimize system performance by reducing the number of page loads and redundant data requests.

4.3.7 Jobs

A `jobs` request retrieves a list of the details of all jobs the user has submitted.

4.3.7.1 Command-Line Application

Listing 4.17 shows the general syntax for the `jobs` request within the command-line application.

Listing 4.17: Jobs Request command structure

```
./raw2rin -r jobs -t TOKEN
```

The only flag is the token:

- `-t`: The token used for user authentication.

Besides the default JSON string previously defined, there doesn't exist any more output for error responses. In a successful response, however, the JSON output includes a `jobs` field, containing an array of job details. Each entry containing the same fields as in the `job` request. For instance, Listing 4.18 shows an example of a successful response of a `jobs` request.

Listing 4.18: Example of a successful response for Jobs Request

```
{
    "jobs":[
        {
            "job_id":80,
            "job_type":"convert",
            "status":"error",
            "err_msg":"The version '1' is not valid. Only versions '2, 3, 4' are allowed
                ↪ for RAW files sbf. Please check the input version and try again."
        },
        {
            "job_id":83,
            "job_type":"convert",
            "status":"completed",
            "err_msg":""
        }
    ],
    "message":"Your jobs have been successfully retrieved!"
}
```

4.3.7.2 Web Application

Figure 4.7 shows the general jobs page of the Web client. For a full-sized image, check Appendix A.1.
The interface includes a table displaying all the user's jobs in the system. Each job is presented with its corresponding ID, type, and status.
If the job has finished processing successfully, a download button will be available to retrieve the results (as explained in Section 4.3.8). Jobs that have not yet finished processing or have not been successful will not display the download button.
If a job has any kind of error, the error message will be displayed instead. Each job also has a delete button that allows the user to schedule its deletion (more details in Section 4.3.9).
The table refreshes its content every 3 seconds via an Asynchronous JavaScript and XML (AJAX) request, ensuring that deleted jobs are removed and the status of ongoing jobs is updated. Users can also manually refresh the table by pressing the refresh button. Additionally, users can schedule the deletion of all jobs at once by using the corresponding button.

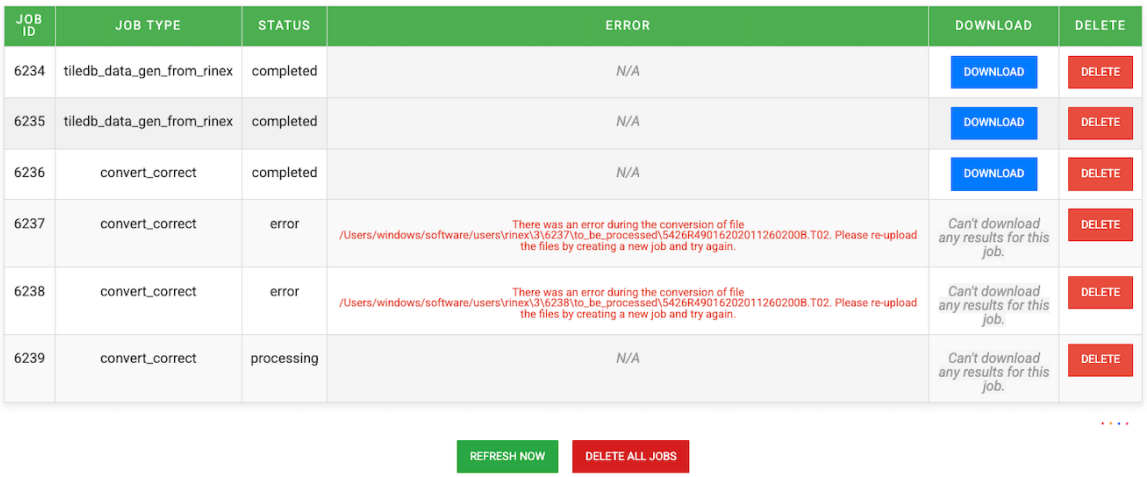


Figure 4.7: The Web client jobs page.

4.3.8 Download

The `download` request allows users to retrieve the output from a specific processing job, such as `convert`, `correct` or `convert_correct`. The format of the output file depends on the type of job performed:

- For `convert` jobs, the output is a single `HATANAKA` compressed RINEX file;
- For `correct` and `convert_correct` jobs, the output is a `tar.gz` file that contains:
  - The resulting RINEX file, compressed in `HATANAKA`;
  - A log file detailing the metadata values that were modified during the correction process;
  - A log file detailing possible warnings, provided by the GFZ software.

4.3.8.1 Command-Line Application

Listing 4.19 shows the general syntax for the `download` request within the command-line application.

Listing 4.19: Download Results command structure

```
./raw2rin -r download -jid JOB_ID -t TOKEN [-o OUTPUT_PATH]
```

The flags are:

- `-jid`: The job's ID of the job to download the results from, obtained from any processing request;
- `-t`: The token used for user authentication;
- `-o`: The path where the results should be downloaded to (default is the current directory).

Upon successful processing of the download results command, the file is retrieved without any additional JSON response. Error responses will follow the default JSON string structure mentioned in section 4.3 without additional information.
The accompanying metadata log file inside the `tar.gz` may look like the one in Listing 4.20 for a change in the `MARKER NAME` only:

Listing 4.20: Example of a log file inside the tar.gz

```
========== Metadata changes for BUMT00BTN_R_20203310000_04H_30S_MO.rnx ==========
OLD LINE: cnvtToRINEX 3.14.0  convertToRINEX OPR  20250119 183621 UTC PGM / RUN BY / DATE
NEW LINE: cnvtToRINEX 3.14.0  convertToRINEX OPR  20250119 183621 UTC PGM / RUN BY / DATE
=====
OLD LINE: BUMT                                                        MARKER NAME
NEW LINE: BUMT00BTN                                                   MARKER NAME
=====
OLD LINE: 0                                                           MARKER NUMBER
NEW LINE: 0                                                           MARKER NUMBER
=====
OLD LINE: GNSS Observer       Trimble                                 OBSERVER / AGENCY
NEW LINE: GNSS Observer       Trimble                                 OBSERVER / AGENCY
=====
OLD LINE: 5426R49016          TRIMBLE NETR9       5.37                REC # / TYPE / VERS
NEW LINE: 5426R49016          TRIMBLE NETR9       5.37                REC # / TYPE / VERS
=====
OLD LINE: 40929390            TRM55971.00     NONE                    ANT # / TYPE
NEW LINE: 40929390            TRM55971.00     NONE                    ANT # / TYPE
=====
OLD LINE:         0.0720        0.0000        0.0000                  ANTENNA: DELTA H/E/N
NEW LINE:         0.0720        0.0000        0.0000                  ANTENNA: DELTA H/E/N
=====
OLD LINE:   3605052.6574  5236537.8562  -510734.0954                  APPROX POSITION XYZ
NEW LINE:   3605052.6574  5236537.8562  -510734.0954                  APPROX POSITION XYZ
=====
============================================
```

4.3.8.2 Web Application

Figure 4.7 shows the download button beside each job. More information can be found in Section 4.3.7.2.

4.3.9 Delete

A `delete` request is used to schedule the deletion of both the input and output of a specific job the user has requested from the system. Notice that this is defined as a scheduled task, meaning it may not be instant. However, note that this action is both non-cancellable and non-reversible.

4.3.9.1 Command-Line Application

Listing 4.21 shows the general syntax for the `delete` request within the command-line application.

Listing 4.21: Delete Request command structure

```
./raw2rin -r delete -jid JOB_ID -t TOKEN
```

The flags are:

- `-jid`: The job's ID of the job to delete, obtained from any processing request;
- `-t`: The token used for user authentication.

Besides the default JSON string previously defined in section 4.3, there doesn't exist any more output for either success or error responses.

4.3.9.2 Web Application

Figure 4.7 shows the delete button beside each job. More information can be found in Section 4.3.7.2.

4.3.10 Clean

Similarly to the `delete` request, the `clean` request is also used to schedule the deletion of both the input and output of jobs, but in this case, of every job issued by the user.

4.3.10.1 Command-Line Application

Listing 4.22 shows the general syntax for the `clean` request within the command-line application.

Listing 4.22: Clean Request command structure

```
1 ./raw2rin -r clean -t TOKEN
```

The only flag is the token:

- `-t`: The token used for user authentication.

Again, besides the default JSON string previously defined in section 4.3, there doesn't exist any more output for either success or error responses.

4.3.10.2 Web Application

Figure 4.7 shows a button for deleting all jobs below the table. More information can be found in Section 4.3.7.2.

4.4 Web API

The Web API receives, routes, and manages client requests through a series of defined endpoints. Each route handles each of the previously defined request types, processes client input collected from Web forms, performs certain validations, and interfaces with the database to store the relevant information.
It runs as a dedicated service on a Web server, and was developed using the Python's Flask framework, which was selected for its lightweight, micro-framework architecture, which requires minimal dependencies, which allows for the integration of custom plugins and modules as needed[FLS]. Its core features–REpresentational State Transfer (REST)ful request

handling, uniform resource locator (URL) routing, and view functions–support the fast prototyping needs of this application[WFL].
Moreover, it is the API which is responsible for including the Web-based client portal that grants users access to all application functionalities through an intuitive browser interface in contrast to the command-line interface. At the time of writing, a public portal can be accessed at https://segal.ubi.pt/raw2rnx.

4.4.1 Routing and Form Handling

Each request type splits into two distinct endpoints: a standard browser endpoint and an API endpoint, with the latter specifically designed for requests originating from the command-line application. While both endpoints utilize the same controller functions, the differentiation is crucial due to the distinct ways in which input is processed by each client. For instance, in the Web client, the user ID is derived from the current user session and passed through routing parameters; conversely, in the command-line application, the user ID must be generated from the user's token, reflecting both the limitations and conventions associated with different client types.
Furthermore, Web client routes are structured to redirect users through their views, rendering the appropriate HTML pages and transmitting information for browser display. In contrast, data intended for the command-line application must be standardized and formatted as JSON to facilitate user automation.
To enhance organization and maintainability, the routes are organized into blueprints, categorizing the application components and supporting common patterns within the application; they serve as a template for constructing and extending the application[BLP].
All input data is processed through form validation utilizing WTForms, a versatile library for form validation and rendering, which provides data validation, data security and internationalization (I18N)[WTF]. Regardless of the source–whether from the Web client or the command-line application–all client data undergoes validation via these forms, despite certain data being captured through different mechanisms. Standard validation checks are implemented for common inputs, such as email and password validation, alongside custom validators tailored to specific requirements, such as verifying that user input for the proprietary format of the conversion jobs adheres to the supported proprietaries.
Additionally, data handling and storage are fortified with robust security measures, including password hashing and salting via the `bcrypt` algorithm[BCR], or protection against common Web vulnerabilities, such cross-site request forgery (CSRF) attacks[CSR].

4.4.2 Job Handling

Most requests rest on the concept of "jobs", which are stored in the database. For each new client request, a unique job entry is created, which the converter application periodically monitors and processes as required. Each job may undergo through a series of well-defined statuses:

- `awaiting`: Applied exclusively to jobs requiring file inputs, indicating that these files

have not yet been uploaded to the converter application server;

- `queued`: The job has been added to the processing queue, where it awaits resource allocation and is processed sequentially, following higher-priority jobs scheduled by the converter application (further details in Section 4.5.5);
- `processing`: The job is currently undergoing active processing by the converter application. Once this job is completed, the next queued job will take its place;
- `completed`: For jobs involving processing, this status signifies that processing is finished and results are ready for the client to download. For other job types, it indicates that all required actions have been successfully executed (for example, user creation);
- `downloaded`: Applied to processing jobs whose results have already been downloaded by the client after completion;
- `error`: An error occurred during job processing, with details accessible to the client for review;
- `error_checked`: Indicates that the client has reviewed the error associated with a job (through either a `job` or `jobs` request).

Certain statuses–namely, `awaiting`, `downloaded`, and `error_checked`–are internal and not directly shown to users. Instead, these statuses are mapped to user-facing equivalents: `awaiting` is displayed as `queued`, `downloaded` appears as `completed`, and `error_checked` is represented as `error`. This distinction allows for consistent user feedback while maintaining the internal state tracking necessary for application functions. For instance, the converter application will only retrieve files from jobs marked as `awaiting` and will remove jobs from the queue once results have been either successfully downloaded or the user has acknowledged an error.
Each job type has specific settings, reflected in the database as various parameters required for processing. For jobs with file inputs, files are initially uploaded to the web server, after which the converter application retrieves them once processing begins; for delete requests, the API also manages the deletion of associated files from the web server, ensuring data consistency with the converter application server.

4.4.3 Database

Figure 4.8 portrays the entity relationship (ER) diagram used by the applications.
Each user is represented by an entry in the `user` table, containing personal information, as well as information required for authentication, viz., `username` or `email` and `password_hash`, along with `token` and `token_expiration` for continuous authentication in command-line clients, i.e., the system generates and stores a token every time the user "logs in" through the command-line application (logging in twice replaces the current token and resets the expiration time) and assigns it an expiration time, after which it stops working and cannot be used anymore to authenticate the clients' requests. The application also implements a quota

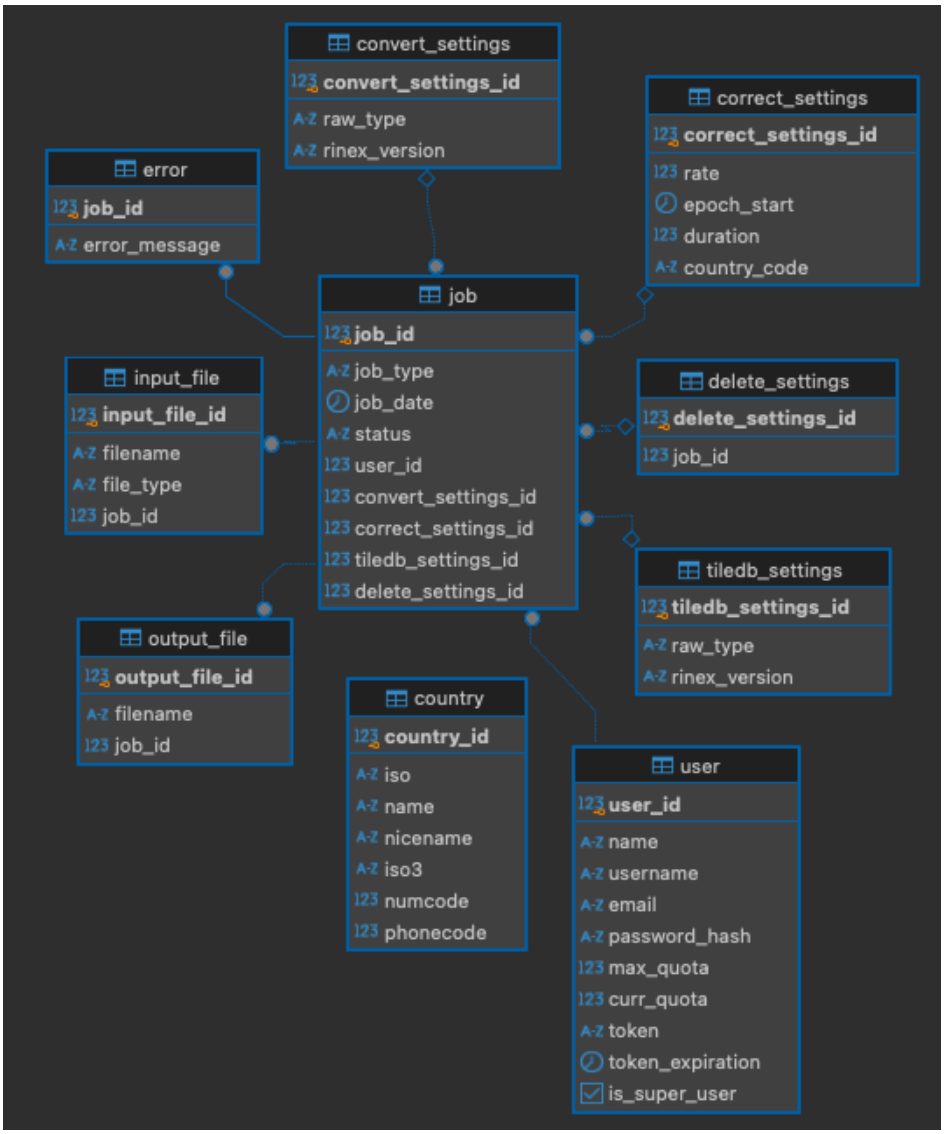


Figure 4.8: Application database ER diagram.

system: each user has a designated `max_quota` that sets their maximum personal storage size. Once a user's current file storage (`curr_quota`) nears this limit, the system restricts additional uploads until existing files are deleted to free up space. By default, each user is allocated a storage quota of 1 GB. Users may also be assigned superuser status (`is_super_user`), granting them elevated privileges, such as higher queue priority for processing requests.

Each user can manage multiple jobs, stored in the `job` table, each configured with job-specific settings depending on the `job_type`. These configurations include `convert_settings`, `correct_settings`, `tiledb_settings` (more on this in Section 5) and `delete_settings`. Additionally, the `status` is used to track the job's current status as previously explained in Section 4.4.2 and `job_date` for scheduling (more on this in Section 4.5.5). Moreover, jobs involving file processing may contain input files (`input_file`), which can be either in RAW form or in the RINEX format, based on the `file_type`; each job's output is stored in `output_files`, making processed results accessible for download.

Error handling is managed via the `error` table, where entries are recorded if a job's `status`

changes to `error`. This table captures error details, enabling users to review and address issues encountered during processing.

4.5 Converter

The `raw2rin` converter application is responsible for handling all processing-intensive jobs related to the conversion of RAW files to RINEX. It operates by periodically querying the database for newly created jobs, which it then submits to a scheduler algorithm, which calculates a priority value for each job and organizes them into a processing queue (further details in Section 4.5.5). Once prioritized, each job is processed individually according to its specific `job_type`.
It is hosted in a cloud environment, similar to the Web server. However, unlike it, which operates on a Linux-based server, `raw2rin` runs on a Windows-based server. This is because, while all the current proprietary conversion software (Septentrio, Trimble, Leica, and ComNav) are designed specifically for Windows, not all of them have Linux-compatible versions; it is technically possible to transition to a Linux-based server in the future, by leveraging technologies such as virtualization, Docker, or Wine, allowing the software to run independently of the operating system (OS).
This software is responsible for updating job statuses, meaning that as jobs are processed or completed, the application updates their statuses in the database, ensuring that the system accurately reflects the current state of each task. Moreover, it manages the deletion of user files, by both removing the specified files from storage and consequently adjusting each user's quota.

4.5.1 File Upload and Validation

For each job that requires file inputs, the initial step is to transfer user-uploaded files from the web server to the converter application server. This transfer occurs securely via secure shell (SSH) between the two servers. Once the files are successfully uploaded, the system updates the user's current quota to reflect the additional storage usage.
Thereafter, the files undergo a thorough validation process based on the user's specified parameters, ensuring that each file meets the necessary requirements before any processing begins. This validation step is essential to confirm that all input files are present and adhere to required formats, versions, and time-related settings, such as duration and sampling rate. By validating these aspects, the system reduces the likelihood of errors occurring during the actual processing, whose ensued faults would be strenuous to catch and handle.

4.5.2 Convert

RAW files are converted **individually** using specialized conversion software, each designed to handle the respective proprietary file format of the raw data. `raw2rin` is flexible in terms of input formats, supporting both compressed (`gzip`) and uncompressed raw files, meaning that if the input files are compressed, the software first uncompresses them before proceeding with the conversion.

Each software tool has its own set of specific settings, which can vary significantly depending on the tool. For instance, some software may allow users to choose the resulting RINEX version, while others do not. Additionally, many software tools provide settings through command-line flags, whereas software like the ComNav conversion software accepts parameters via the standard input. As such, each conversion tool requires individual handling to ensure compatibility with its specific behavior.
Some proprietary conversion software **solely** generate RINEX observation files, while others produce a broader range of output, including observation, navigation, and even meteorological files. In cases where the software generates additional files beyond the observation data, these are discarded as they are not needed for the intended conversion output.
The resulting RINEX files are placed in unique folders, each named with a unique SHA512 hash, pertaining to the original filename, the conversion date, and a secret key, ensuring that the likelihood of hash collisions is minimized. At that point, they are merged together using the GFZ software, which assigns the 9-character long name associated to the station of the file, following the standard naming convention for RINEX files.
If no errors occur during these processes, the conversion is considered successful. The merged RINEX file is, thus, compressed using the HATANAKA software and moved to the processed directory, where they are made available for retrieval by the user.

4.5.3 Correct

All corrections made to the RINEX files are handled through the GFZ software. Some corrections are applied directly as flags within the software, such as adjusting the sampling rate or epoch. However, metadata corrections require a more structured approach, which is facilitated through a specific configuration file format called CRUX. This configuration file is designed to edit and manipulate the headers of RINEX files systematically, providing a structured way to apply specific editing operations, including updating, inserting, or replacing header lines[CRU].
The CRUX file utilizes a line-by-line syntax that defines how and when changes should be made. An example of a CRUX file is provided in Listing 4.23.

Listing 4.23: A CRUX file.

```
1 update_insert:
2 ALL:
3     "REC # / TYPE / VERS" : { 0: "4831K57521", 1: "TRIMBLE NETR5", 2: "Nav 4.87 / Boot
      ↪ 4.18"}
4     "ANT # / TYPE"        : { 0: "30767802",   1: "TRM55971.00"}
```

In this example, the CRUX file instructs the software to either update or insert metadata for all data types (observation, navigation, and meteorological) across all epochs. The specific headers targeted for change are REC # / TYPE / VERS and ANT # / TYPE and the values within the CRUX file represent key-value pairs that specify the location of the metadata within the RINEX header, with the corresponding value indicating the new data to be inserted or updated; for instance the REC # field under the header REC # / TYPE / VERS is updated to the value 4831K57521, because, as shown in Figure 4.9, the number 0 corresponds to the

first parameter of this specific header. The actual values to be modified in the RINEX file come from the JSON file mentioned in section 4.3.4.

Figure 4.9: Metadata correspondence.

Once the metadata corrections are applied, a log file is created documenting the changes made to the RINEX metadata, which shows which pieces of metadata were altered, indicating the original and new values for each field. Additionally, the GFZ software generates its own log during the correction process. Both the CRUX correction log and the GFZ software log are bundled together alongside the corrected RINEX file converted to HATANAKA, so that the `tar` resulting file is compressed using `gzip` and placed in the `processed` directory, ready for retrieval by the user.
In cases where files undergo both conversion and correction jobs, the output from the conversion job is passed directly to the correction process without first being compressed into the `HATANAKA` format. The corrector is designed to accept uncompressed RINEX files as well as files compressed using both or any of the following–`gzip` or `HATANAKA`–regardless of the order in which they are compressed.
During the correction process, the corrector attempts to derive the standard 9-character filename for the resulting RINEX file. If the initial file lacks sufficient information to generate this filename, the following checks are performed:

- Station Name: The corrector verifies whether a valid 4-character station name exists in the `marker` field of the settings file. If found, it replaces the default `XXXX` with the appropriate station name;
- Country Code: The corrector examines the `long_marker` field in the settings file for the last three characters of the country code. If this field is absent, it looks for the country code parameter specified in the request.

Using these two pieces of information, the corrector generates the final, standardized 9-character filename for the RINEX file.

4.5.4 File System

Upon user registration, the converter application allocates dedicated file system space for the new user designated for storing all RINEX file results. Within it, a unique subdirectory is created for each user, named according to their user ID.
Within each ID directory for a job, files awaiting processing are stored in a designated `to_be_processed` folder. Here, all active processing takes place; once completed, the resulting files are saved to a `processed` directory within the job's structure, where they await user retrieval.
Listing 4.24 illustrates an example of the system's directory structure.

Listing 4.24: Filesystem example

```
Users/
|-- RINEX/
    |-- 1/
    |   |-- 789/
    |   |   |-- to_be_processed/
    |   |   |-- processed/
    |   |-- 791
    |       |-- to_be_processed/
    |       |-- processed/
    |-- 2/
    |-- 3/
        |-- 790/
            |-- to_be_processed/
            |-- processed/
```

According to the example, the system currently supports three users with IDs ranging from 1 to 3. User 1 has two active jobs (IDs 789 and 791), user 2 has no active jobs, and user 3 has one job (ID 790).

4.5.5 Scheduling

The converter software employs a scheduling mechanism that regularly reorders the jobs in a queue to ensure that they are processed in an order that prioritizes critical tasks while optimizing fairness and resource usage.
The process begins by analyzing a list of jobs that need processing. Each job is assigned a priority score, which reflects its importance relative to others in the queue. High-priority jobs, such as user creation or deletion, are given precedence. Once priorities are calculated, the jobs are sorted in descending order of priority. If two or more jobs have the same priority, they are further sorted by their unique job identifiers.
Critical jobs of certain types are assigned fixed, predefined priority scores above 1 to ensure they are always prioritized over non-crucial jobs, whereas for other regular processing jobs, the priority is dynamically computed using the formula in equation 4.1.

$$\text{Priority Score} = \text{Waiting Time Score} \times \gamma_1 + \text{Usage Score} \times \gamma_2 \quad (4.1)$$

The waiting time score measures how long a job has been in the queue, meaning that jobs that have been waiting longer are assigned higher scores. It is calculated based on the difference between the current time and the job's submission time, as shown in equation 4.2.

$$\text{Waiting Time Score} = \min\left(\frac{\text{Current Time} - \text{Job Submission Time}}{\text{MAX\_WAITING\_TIME}}, 1\right) \quad (4.2)$$

The `MAX_WAITING_TIME` is defined as 60 minutes on this system.
The usage score evaluates how many jobs a user has submitted recently. In order to prevent any single user from monopolizing the system, the system tracks the number of jobs submitted

by each user over a recent period (viz., the last 24 hours) and users with a higher submission count receive lower scores, reducing the priority of their jobs, as shown in equation 4.3.

$$\text{Usage Score} = \min\left(\max\left(\frac{\text{MAX\_USER\_USAGE\_JOBS} - \text{Jobs Submitted}}{\text{MAX\_USER\_USAGE\_JOBS}}, 0\right), 1\right) \quad (4.3)$$

The `MAX_USER_USAGE_JOBS` is defined as 5 minutes on this system.
Both the `Waiting Time Score` and the `Usage Score` are normalized between 0 and 1; this ensures that the scores are positive, with 0 representing the minimum possible score and 1 representing the maximum. As a result, and since critical jobs have fixed priorities greater than 1, regular processing jobs will never be scheduled ahead of the latter. Figures 4.10 and 4.11 illustrate the functions used to calculate the `Waiting Time Score` and the `Usage Score`, respectively.

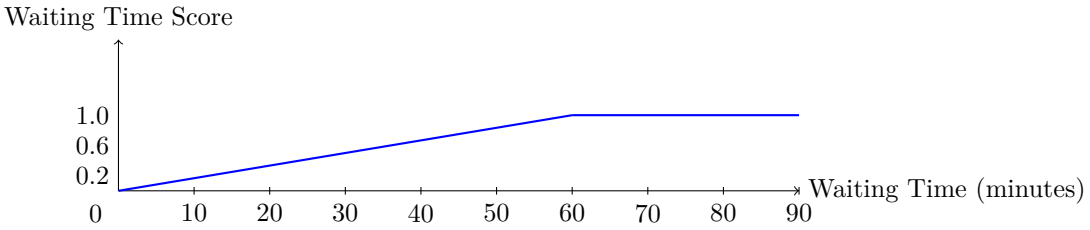


Figure 4.10: Waiting time score graph.

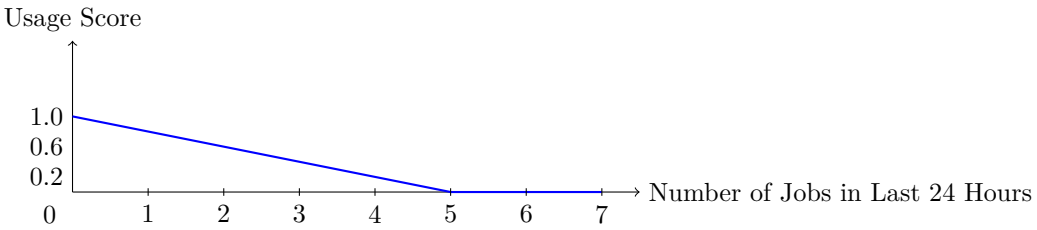


Figure 4.11: Usage score graph.

Additionally, the parameters γ_1 and γ_2, shown in equation 4.1, represent the weights of the `Waiting Time Score` and the `Usage Score`, respectively. The constraint $\gamma_1 + \gamma_2 = 1$ and $0 \leq \gamma_1 \leq 1, 0 \leq \gamma_2 \leq 1$ ensures that the total weight is divided between the two scores in a manner that reflects the system's specific needs. Right now, they are both defined to be equal to 0.5, meaning they have the same weight, but further testing of the system may reveal that one may need to be greater than the other.
To illustrate the scheduling mechanism, consider a queue of six jobs submitted by three different users. The details of the jobs are shown in table 4.1.
Assuming $\gamma_1 = \gamma_2 = 0.5$, the calculated priorities are shown in Table 4.2.
Notice that the jobs with IDs 2 and 8 have the same priority score (0.5). In such cases, the jobs are sorted by their unique job identifiers in ascending order, therefore, the job with ID 2 precedes the one with ID 6 in the final job queue. The final sorted job order is:

$$\text{Job Order: } \mathbf{6} \rightarrow \mathbf{1} \rightarrow \mathbf{9} \rightarrow \mathbf{7} \rightarrow \mathbf{5} \rightarrow \mathbf{2} \rightarrow \mathbf{8} \rightarrow \mathbf{3} \rightarrow \mathbf{4}.$$

Job ID	User	Waiting Time (min)	User Job Count (Last 24h)	Job Type
1	User A	0	4	Job Deletion
2	User B	0	0	Data Processing
3	User B	5	1	Data Processing
4	User A	30	5	Data Processing
5	User C	45	0	Data Processing
6	User B	60	2	User Creation
7	User C	60	1	Data Processing
8	User A	80	6	Data Processing
9	User D	90	8	Super User Data Processing

Table 4.1: Initial list of jobs in the queue.

Job ID	Waiting Time Score	Usage Score	Priority Score
1	N.A.	N.A.	3
2	0	1	0.5
3	0.08	0.8	0.44
4	0.5	0	0.25
5	.75	1	0.875
6	N.A.	N.A.	4
7	1	0.8	0.9
8	1	0	0.5
9	N.A.	N.A.	2

Table 4.2: Calculated priority scores for jobs.

4.6 Conclusion

This chapter outlined the architecture and key functionalities of the software package. Beginning with the client interfaces—the command-line and web applications—the system provides users with a flexible and accessible way to initiate jobs, manage files, and retrieve results. Input validation and quota management ensure data integrity and system resource balance, preventing errors and securing user data.

The web API and the converter application play central roles in data management and processing, respectively. The first serves as the gateway, handling data validation, user authentication, and routing client requests to the appropriate endpoints. Each request is then stored as a job within the database. The converter, then, organizes and prioritizes each job and consequently, processes all specific tasks, including data conversion, corrections, and job status management.

The conversion software handles proprietary formats, transforming raw data files into standardized RINEX files, while metadata adjustments and corrections are made via the GFZ's `gfz2rnx` software with custom metadata configurations managed through CRUX files. Final outputs are stored in user-specific directories on the server for organized retrieval.

Chapter 5

TileDB Data Generation and Storage Using TileDB Manager

5.1 Introduction

Originally, the thesis aimed to evaluate the performance of TileDB, so while the focus has since shifted toward the raw2rin API, assessing TileDB's performance remains an important objective. To facilitate this, it is crucial to extend the overall API to support the import of RINEX data into TileDB.
This chapter will, therefore, concentrate on the bidirectional conversion between RINEX and TileDB, enabling both the import of RINEX data into TileDB and the generation of RINEX files from it for further use.

5.2 TileDB Importable Data

It is straightforward to convert Python's Pandas' DataFrames to TileDB format and vice versa. Hence, the first step of importing the data into TileDB is to parse the observation data from RINEX files into a Pandas DataFrame, which is facilitated by GFZ's Pynex[Brab] software, which has been adapted to meet the specific needs of this software.

5.2.1 TileDB Data Generation From Raw

A tiledb_data_gen_from_raw request is used to convert one or more RAW files into a single RINEX file and then to a Pandas DataFrame.
After the conversion explained in Section 4.5.2, Pynex reads and parses the resulting RINEX file, extracting all data (time, system, satellite, and various observation types (e.g., phase, pseudorange, Doppler, SNR)) and compiling it into a Pandas DataFrame. The resulting DataFrame is written onto a comma-separated values (CSV) file, which is compressed and returned once the user downloads the result.

5.2.1.1 Command-Line Application

Listing 5.1 shows the general syntax for the tiledb_data_gen_from_raw request within the command-line application.

Listing 5.1: Tiledb Data Generation From Raw command structure

```
./raw2rin -r tiledb_data_gen_from_raw -pf PROPRIETARY_FORMAT -v VERSION -i INPUT_FILES…, -t
    ↪ TOKEN [-np] [-pr]
```

The flags are the same present in the convert request of Section 4.3.3.
Listing 5.2 shows an example of a tiledb_data_gen_from_raw request command.

Listing 5.2: Example of a Tiledb Data Generation From Raw Request command

```
./raw2rin -r tiledb_data_gen_from_raw -pf sbf -v 3 -i "BUMT202012191700B.sbf.gz" -t
    ↪ f3518eef004568f34eb973466504ab2c
```

Upon successful processing of the request, the response returns a JSON message containing a job ID (more on this in section 4.4.2), which can be used to check the status of the job and download its results after processing. For error responses, only the standard error message structure mentioned in section 4.3 is returned.
Listing 5.3 shows an example of a tiledb_data_gen_from_raw success response.

Listing 5.3: Example of a successful Tiledb Data Generation From Raw response

```
{
    "job_id": 91287,
    "message": "Your files have been queued for upload!"
}
```

5.2.2 TileDB Data Generation From RINEX

A tiledb_data_gen_from_rinex request is used to convert one RINEX file to a Pandas DataFrame.
Unlike the correct request, this request does not allow any kind of customization or metadata changing; its only purpose is to generate the importable Pandas' DataFrame compressed CSV file and not to generate RINEX files.

5.2.2.1 Command-Line Application

Listing 5.4 shows the general syntax for the tiledb_data_gen_from_rinex request within the command-line application.

Listing 5.4: Tiledb Data Generation From RINEX command structure

```
./raw2rin -r tiledb_data_gen_from_rinex -i INPUT_FILE -t TOKEN [-np] [-pr]
```

The flags are the same present in the correct request of Section 4.3.4.
Listing 5.5 shows an example of a tiledb_data_gen_from_rinex request command.

Listing 5.5: Example of a Tiledb Data Generation From RINEX Request command

```
./raw2rin -r tiledb_data_gen_from_rinex -i BUMT00BTN_R_20203541700_01H_30S_MO.rnx -t
    ↪ f3518eef004568f34eb973466504ab2c
```

Upon successful processing of the request, the response returns a JSON message containing a job ID (more on this in section 4.4.2), which can be used to check the status of the job and download its results after processing. For error responses, only the standard error message structure mentioned in section 4.3 is returned.
Listing 5.6 shows an example of a tiledb_data_gen_from_rinex success response.

Listing 5.6: Example of a successful Tiledb Data Generation From RINEX response

```
{
    "job_id": 91289,
    "message": "Your files have been queued for upload!"
}
```

5.3 TileDB Manager

While the `tiledb_data_gen_from_raw` and `tiledb_data_gen_from_rinex` requests focus on generating data for import into TileDB, and make use of the `raw2rin` converter–particularly for converting RAW files into RINEX in the case of the `tiledb_data_gen_from_raw` request– this integration is sensible within the context of the `raw2rin` command-line application.
However, the actual tasks of importing RINEX data into TileDB and generating RINEX files should be handled separately. Therefore, a new command-line application was created, called `tiledb_manager`. It functions similarly to `raw2rin`, but exclusively supports these TileDB-related tasks and implements some code of [Braa].

5.3.1 Insert into TileDB

An `insert_into_tiledb` request is a local request and is used to import the DataFrame data inside the compressed CSV file into the user's designated TileDB storage space.
The TileDB storage path is dynamically generated based on the station name and date (year, month, day) extracted from the CSV filename[1], with the time granularity serving as a key factor in how the data is organized. The chosen granularity (e.g., station-only, station and year, or station, year, and month) determines how the data is segmented across different TileDB arrays. For instance, if the time granularity is set to year, a separate TileDB array will be created for each station-year combination. At this stage, the exact granularity that works best is not yet determined, and it can be modified as needed to optimize the data storage and access for specific use cases.
If no TileDB array exists for the specified station and time granularity, a new one is created according to a schema that is similar to that used by UNAVCO (referred to in section 3.3.0.1) and GFZ.
The chosen schema contains the following TileDB dimensions:

- Time Dimension (`time`) — The different observation epochs:
  - Data Type: `np.int64` (64-bit integer representing miliseconds);
  - Domain: From 315964800000 milliseconds since the GPS epoch, corresponding to January 6, 1980, to 4102444800000 milliseconds, corresponding to January 1, 2100;

[1]Note that the CSV files returned by both the DataFrame generation requests are named according to the 9 character long filename of the RINEX files they derive from with the addition of the CSV and gzip file extensions, meaning that it is possible to derive the station name and date from them.

  - Tile Size: 6 hours (21600000 milliseconds).

- System Dimension (`sys`) — The different systems associated with each observation:
  - Data Type: `ascii` (A single-character system code: G: GPS, R: GLONASS, S: SBAS Payload, E: Galileo, C: BeiDou, J: QZSS, I: NavIC).
- Satellite Dimension (`sat`) — The satellite identification number (PRN code or slot number).
  - Data Type: `np.uint8` (8-bit unsigned integer).
- Observation Type Dimension (`obs`) — The actual observation.
  - Data Type: `ascii` (The observation types).

The attributes define the types of data associated with each observation:

- Pseudorange (`p_range`):
  - Data Type: `np.float64` (64-bit floating point).
- Carrier Phase (`phase`):
  - Data Type: `np.float64` (64-bit floating point).
- Doppler Shift (`doppler`):
  - Data Type: `np.float64` (64-bit floating point).
- Signal-to-Noise Ratio (`snr`):
  - Data Type: `np.float32` (32-bit floating point).
- Loss-of-Lock Indicator (`lli`):
  - Data Type: `np.uint16` (16-bit unsigned integer).
- Signal Strength Indicator (`ssi`):
  - Data Type: `np.int16` (16-bit signed integer).
- Flags (`flags`) — Empty for now, but for future-proofing as the RINEX format evolves.
  - Data Type: `np.uint16` (16-bit unsigned integer).

The array is sparse because not all satellites are observed at all times, so empty cells are allowed, which allows for much more efficient storage; both the cell order and tile order are defined as `row-major`, meaning the array is stored with the fastest-changing dimension being the time dimension, followed by the system, satellite, and observation dimensions. This is imperative, because most queries will be based on time.

All the dimensions and attributes are compressed using different compression filters, such as `ZstdFilter` and `BitWidthReductionFilter`; these filters and some code used to populate the TileDB was taken from [Braa].
After the TileDB structure is in place, the observation data from the DataFrame is directly imported to the TileDB array.

5.3.1.1 Command-Line Application

Listing 5.7 shows the general syntax for the `insert_into_tiledb` request within the command-line application.

Listing 5.7: Insert into TileDB command structure

```
./tiledb_manager -r insert_into_tiledb -td TILEDB_DIR -ods OBSERVATION_DATAFRAME_FILE [-np]
    ↪ [-pr]
```

It contains the following flags:

- `-td`: Either the current user's TileDB directory or an empty one, where a TileDB directory will be created;
- `-ods`: The CSV file containing the DataFrame observation data to be imported into TileDB. It must have a 9-character long filename, followed by the `.csv` extension, as it contains the necessary information to derive the station, year, month, and day, which are then used to form the TileDB granularity storage path.

Listing 5.8 shows an example of a `insert_into_tiledb` request command.

Listing 5.8: Example of an insert into TileDB Request command

```
./tiledb_manager -r insert_into_tiledb -td user/tiledb_dir -ods
    ↪ BUMT00BTN_R_20203541700_01H_30S_MO.rnx.csv
```

No meaningful response is returned from this request.

5.3.2 Add Metadata

A `add_metadata` request is also a local request and is used to import the required metadata to a local database from an international GNSS system (IGS)-type log file, so that it is possible to regenerate the RINEX file back from the TileDB.
Most metadata is not imported to the TileDBs, and it is not feasible to first import the RINEX metadata into a local SQLite3 database, as some of this metadata may change over time.
Figure 5.1 illustrates the ER diagram of the local database used to store the metadata required to regenerate the RINEX file. The database only includes the mandatory metadata that is not already implied by the TileDB data (viz., `SYS / # / OBS TYPES` , `RINEX VERSION / TYPE`, `TIME OF FIRST OBS` and `TIME OF LAST OBS`) or generated artificially (viz., `PGM / RUN BY / DATE` and `END OF HEADER`).

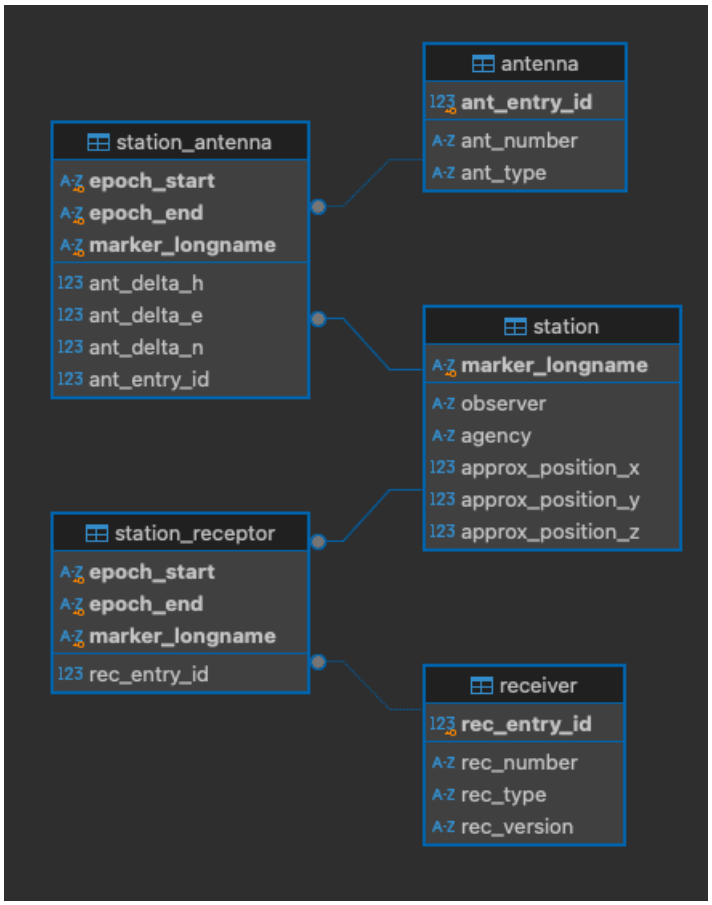


Figure 5.1: Local metadata database ER diagram.

The database contains a station table that stores specific metadata about the station. Additionally, it tracks a history of the different receivers and antennas used at each station, spanning a specific period, to reflect the changes in equipment over time.
Given the local database path, the database is created if it doesn't already exist. Once the database is in place, metadata can be retrieved from an IGS-type log file corresponding to the specific station from which data is to be imported. This file contains several sections: one section with the station's metadata, and two additional sections that provide the history of the receiver and antennas used at that station (see Appendix A.1 for a template of an IGS-type log file.
The only two mandatory and non-implicit metadata parameters that aren't present in the IGS-type log files are the observer and agency, so those need to be explicitly given as input on the request.

5.3.2.1 Command-Line Application

Listing 5.9 shows the general syntax for the `add_metadata` request within the command-line application.

Listing 5.9: Add metadata command structure

```
./tiledb_manager -r add_metadata -md METADATA_DIR -lf IGS_LOG_FILE -ob OBSERVER -ag AGENCY
    ↪ [-np] [-pr]
```

It contains the following flags:

- `-md`: Either the current user's metadata directory or an empty one, where the local metadata database will be created;
- `-lf`: The IGS-type log file;
- `-ob`: The actual observer metadata parameter;
- `-ag`: The actual agency metadata parameter.

Listing 5.10 shows an example of an `add_metadata` request command.

Listing 5.10: Example of an add metadata Request command

```
./tiledb_manager -r add_metadata -md user/metadata_dir -ods bumt_20230816.log -ob C4G -ag
    ↪ SEGAL
```

No meaningful response is returned from this request.

5.3.3 Generate RINEX

A `generate_rinex` request is also a local request and is used to regenerate the RINEX file using the data stored in the user's TileDB, the metadata stored in the user's local metadata directory, according to certain parameters. This request is analogous to a *query* in a database. Therefore, from now on, whenever a query to TileDB is mentioned, it should be understood as referring to this request.
To retrieve data from TileDB and reconstruct a RINEX file, the process is designed to be precise, especially when working with time-related information. Because the TileDB data can be organized based on different time granularities, a query may require data that spans across multiple TileDB arrays, depending on the selected epoch period.
When querying for time-based data and satellite-specific information, the system uses specific query parameters to ensure that only the relevant data is retrieved from TileDB.
Once the necessary data is collected into the DataFrame, the Pynex library is used to generate the RINEX file. To make sure the RINEX file contains all the required information, a query is made to the local metadata database to fetch any missing metadata, based on the station and the time period of interest. This metadata is then combined with the observation data in the DataFrame to produce the final RINEX file, ensuring that both the data and metadata are accurately integrated.

5.3.3.1 Command-Line Application

Listing 5.9 shows the general syntax for the `generate_rinex` request within the command-line application.

```
./tiledb_manager -r generate_rinex -td TILEDB_DIR -md METADATA_DIR -o OUTPUT_PATH -st
    ↪ STATION -rst RINEX_START_TIME -ret RINEX_END_TIME -int INTERVAL [-gsat] [-rsat] [-
    ↪ esat] [-csat] [-jsat] [-ssat] [-isat] [-np] [-pr]
```

Listing 5.11: Generate RINEX command structure

It contains the following flags:

- `-td`: The TileDB directory where data is stored;
- `-md`: The local metadata directory used to store the metadata database;
- `-o`: The output path where the generated RINEX file will be saved;
- `-st`: The station 9 character long filename, specifying the station for the data extraction;
- `-rst`: The start time for the RINEX file, indicating the beginning of the time period for the data to be included;
- `-ret`: The end time for the RINEX file, indicating the end of the time period for the data to be included;
- `-int`: The interval between consecutive observations in the RINEX file, determining the frequency of the data entries;
- `-gsat`: Do not include the global positioning system (GPS) satellite system;
- `-rsat`: Do not include the globalnaya navigazionnaya sputnikovaya sistema (GLONASS) system;
- `-esat`: Do not include the Galileo system;
- `-csat`: Do not include the BeiDou system;
- `-jsat`: Do not include the quasi-zenith satellite system (QZSS) system;
- `-ssat`: Do not include the satellite based augmentation system (SBAS)system;
- `-isat`: Do not include the navigation with indian constellation (NavIC) system.

Listing 5.12 shows an example of a `generate_rinex` request command.

Listing 5.12: Example of a generate rinex Request command

```
./tiledb_manager -r generate_rinex -td user/tiledb_dir -md user/metadata_dir -o user/
    ↪ output_rinex -st BUMT00BTN -rst 2020-01-01_00:00 -ret 2020_01-02_00:00 -int 30 -rsat
```

No meaningful response is returned from this request.

5.4 Conclusion

This chapter demonstrates the critical processes involved in converting and importing RINEX data into TileDB. By using both the `raw2rin` and the `tiledb_manager` applications, it is enabled the integration of RINEX datasets into TileDB, making them accessible for further

analysis. The detailed explanation of data generation from both RAW and RINEX files into TileDB-compatible formats, alongside the insertion of data into TileDB storage, illustrates the practical aspects of working with large GNSS datasets, providing a robust system for handling, storing, and accessing GNSS observation data efficiently within TileDB, creating the framework for the tests conducted in chapter 6.

Chapter 6

TileDB Evaluation

6.1 Introduction

This chapter outlines the tests conducted to assess the performance of the TileDB-based system developed for storing and querying GNSS RINEX data. The evaluation primarily focuses on the impact of different storage granularities (i.e., creating one TileDB for each station (station-only granularity), each station-year pair (station-year granularity), each station-year-month combination (station-year-month granularity). While these tests do not encompass the full spectrum of potential TileDB configurations, they may provide some insights into the performance of TileDB.

6.2 System Environment

The tests were conducted on a virtual server with the following hardware:

- CPU: Intel(R) Xeon(R) Silver 4112 CPU @ 2.60GHz (4 cores);
- Virtual memory size: 3.8GiB;
- Storage type: SSD.

It has the following software features:

- Operating system: Ubuntu 22.04.4 LTS;
- TileDB version: 0.33.0;
- Python version: 3.10.16.

6.3 Data Preparation

The RINEX files used for testing came from three GNSS stations (CAS100ATA, TLSE00FRA, MAS100ESP) and spanned a period of 11, 9, and 12 years, respectively.
The chosen timeframe for processing the files was based on the objective gathering of all available data from SEGAL; the reasoning behind choosing these particular three stations was somewhat arbitrary; however, they were chosen because they contained a substantial amount of data, covering a lot of years in the dataset. Note that, due to the long time span, not all years and months had complete data coverage, as some days were missing.
The data preparation process began by converting the RINEX files into DataFrames automatically by use of a custom-built script. This process took around three days for the 9151

different files. These DataFrames were stored in compressed CSV files, making it easier to manage and manipulate the data. The script iterated through the yearly, monthly, and daily directories containing the RINEX files, invoking the command-line application to process each file individually and automatically and return its corresponding results; the result was a series of compressed CSV files ready to import into TileDB.

Once they were prepared, they were ingested into TileDB. The ingestion process was conducted three times (to get a statistically significant import timing value[1]) for each of the three granularities separately, namely, for the station-only granularity, all data for a given station was stored in a single TileDB array, for the station-year granularity, data was divided into yearly partitions, creating a separate TileDB array for each station-year combination, and for the station-year-month granularity, the data was further divided by month as well. The resulting file system can be observed in Listing 6.1.

Listing 6.1: Test data file system example

```
tiledb_archive/
|-- tiledb_station_only/
|   |-- CAS100ATA/ (TileDB)
|   |   |-- __commits/
|   |   |   |-- ...
|   |   |-- __fragment_meta/
|   |   |   |-- ...
|   |   |-- __fragments/
|   |   |   |-- ...
|   |   |-- .../
|   |-- MAS100ESP/ (TileDB)
|   |   |-- ...
|   |-- TLSE00FRA/ (TileDB)
|       |-- ...
|-- tiledb_station_year/
|   |-- CAS100ATA/
|   |   |-- 2014/ (TileDB)
|   |   |   |-- __commits/
|   |   |   |   |-- ...
|   |   |   |-- ...
|   |   |-- 2015/ (TileDB)
|   |   |   |-- __commits/
|   |   |   |    |-- ...
|   |   |   |-- ...
|   |   |-- ...
|   |-- ...
|-- tiledb_station_year_month/
    |-- CAS100ATA/
    |   |-- 2014/
    |   |   |-- 4/ (TileDB)
    |   |   |   |-- __commits/
    |   |   |   |   |-- ...
    |   |   |   |-- ...
    |   |   |-- 5/ (TileDB)
    |   |   |   |-- __commits/
    |   |   |   |   |-- ...
    |   |   |   |-- ...
    |   |   |-- ...
    |   |-- ...
    |-- ...
```

[1]No data was repeated in the TileDBs, however more fragments were generated; this was accounted to.

6.4 Performance Metrics

The evaluation was based on multiple performance metrics, selected to provide a comprehensive understanding of the system's behavior.
For the query operations, the following metrics were chosen:

- Execution time (s): the duration required to execute each query;
- Memory usage (GB): the total memory consumption during each query;
- central processing unit (CPU) utilization (%): The average processing power required during each query.

For the import operations, only the execution time metric was measured. While execution time is the primary focus for optimizing the overall system performance, the remaining metrics also provide valuable insights into the system's efficiency, as they may offer a more comprehensive understanding of how the system operates under various conditions and how resources are being utilized during processing.
All the metrics were computed using the Linux `time` command. The execution time is the real time value.The memory usage is the resident set size (RSS)[Ora10] value. The CPU utilization was computed using the formula in Equation 6.1[Mad24][Tec24]; note that, because the CPU has four cores, this must be accounted for in the formula, so the `nproc` command was used to get the number of available cores as well.

$$\text{CPU Usage (\%)} = \left(\frac{\text{User CPU Time} + \text{System CPU Time}}{\text{Elapsed Time} \times \text{Number of CPU Cores}} \right) \times 100 \quad (6.1)$$

Moreover, the storage size and fragment number of each granularity is also chosen to contrast against the other metrics.

6.5 Test Scenarios

The tests were designed to cover a range of realistic query scenarios. For the query operations, the first set of tests random daily queries, daily queries for a whole month, and queries spanning two days. Import operations were simpler to evaluate, as they involved measuring the time and storage requirements for ingesting all data into each granularity.

6.6 Results

The results of the evaluation are presented in the following sections, divided by granularity for clarity.

6.6.1 Write Performance

The results of the compressed CSV files' import are shown in Table 6.1. The imports were performed three times, and the averages were computed to ensure statistical significance.

Granularity	Average Processing Time (hours)	Number of TileDBs	Total Size (GB)
TileDB per station only	11.47h	3	114GB
TileDB per station, and year	11.50h	32	114GB
TileDB per station, year, and month	11.44h	314	114GB

Table 6.1: Import times for different granularities for 9151 total different files.

6.6.2 Read Performance

For several days, and for each of the three stations, five different queries spanning one day were conducted ten times to account for statistical significance for each of the three granularities. The results are presented in tables A.1, A.2, and A.3, corresponding to the station-only, station-and-year, and station-year-month granularities, respectively in Appendix A.3.

For a whole month, 31 different queries spanning one day were conducted five times to account for statistical significance for each of the three granularities with and without consolidation. The results are presented in tables A.4, A.5, and A.6, corresponding to the station-only, station-and-year, and station-year-month granularities, respectively in Appendix A.3. Note that all fragments are equal for each day, being 11520, 1089, and 93 fragments for the station-only, station-and-year, and station-year-month granularities, respectively.

For three periods, three different queries spanning two days between two different months and years were conducted five times to account for statistical significance for each of the three granularities with and without consolidation. The results are presented in tables A.7, A.8, and A.9, corresponding to the station-only, station-and-year, and station-year-month granularities, respectively in Appendix A.3.

6.7 Discussion

The import process for the three granularities took a considerable amount of time due to the large volume of imported data. However, as the number of TileDB instances increased, the average processing time did not exhibit a noticeable rise, suggesting that the creation of TileDBs is nearly instantaneous. Additionally, the number of fragments did not appear to impact write performance, as previously discussed in [Til24].

Regarding storage size, the addition of extra TileDBs did not lead to a significant increase in overall size, with all instances maintaining approximately the same size.

Tables 6.2, 6.3, and 6.4 show the variance of performance between granularities for execution time, CPU usage, and memory usage, respectively.

Granularity	Avg Time (s)	Faster than Station Only (%)
Station Only	68.84	-
Station-Year	40.25	41.5%
Station-Year-Month	39.38	42.8%

Table 6.2: Time Comparison Between Granularities

Granularity	Avg CPU Usage (%)	Increase from Station Only (%)
Station Only	25.53%	-
Station-Year	30.65%	+20.1%
Station-Year-Month	31.07%	+21.7%

Table 6.3: CPU Usage Comparison Between Granularities

Granularity	Avg Memory (GB)	Lower than Station Only (%)
Station Only	0.768	-
Station-Year	0.678	11.7%
Station-Year-Month	0.675	12.1%

Table 6.4: Memory Usage Comparison Between Granularities

Figure 6.1 presents a grouped bar chart comparing the average values of the three performance metrics across the three granularities for daily queries.

Analyzing the graphs and tables, a clear pattern emerges: the fewer TileDBs used to store the data, the longer the query execution time. This effect is particularly evident when comparing the station-only TileDB configuration to the monthly and yearly databases, i.e., the difference in query times between monthly and yearly configurations is relatively small, but the increase in execution time becomes substantial in the station-only case. This suggests that the variance is not linear, as also shown in the variance figures.

Figure 6.2 shows the variance in execution time across all tested days; again, as can be seen, the station-year is close to the year-month, while the station only stands out as the most varying of the three, pertaining to a worse performance.

An interesting observation is that CPU usage percentage decreases as the number of TileDBs decreases; while the decrease is relatively minor between the yearly and monthly databases, there is a significant drop in CPU utilization for the station-only configuration. This suggests that, when using fewer but larger TileDBs, such as in the station-only configuration, the lower CPU usage could indicate that the queries are bottlenecked by disk I/O, leading to longer overall execution times. Conversely, when using more but smaller TileDBs, such as in the monthly or yearly configurations, the higher CPU usage suggests that more of the processing happens in memory, reducing reliance on slower disk reads. The station-only configuration contains many fragments within a single database, which increases the time required to scan and retrieve data. Instead of parallelizing computation across multiple TileDBs, the system may be forced to wait on disk operations more frequently, leading to lower CPU utilization.

Memory usage for read queries remains relatively low, consistently under 1GB. While memory consumption does increase slightly as the number of fragments grows, this increase does not appear to be statistically significant between the yearly and monthly configurations.

One possible conclusion from this is that the number of fragments positively correlates with query execution time and total memory usage, while it inversely correlates with CPU utilization. However, this correlation is not absolute–occasional deviations suggest that hardware factors such as caching, disk I/O speeds, or parallel processing capabilities may influence the results.

Higher memory usage and execution times also appear to correlate with the volume of queried data and the resulting file sizes. For example, the generated files CAS100ATA_R_20220660000_01D_30S_MO.rn

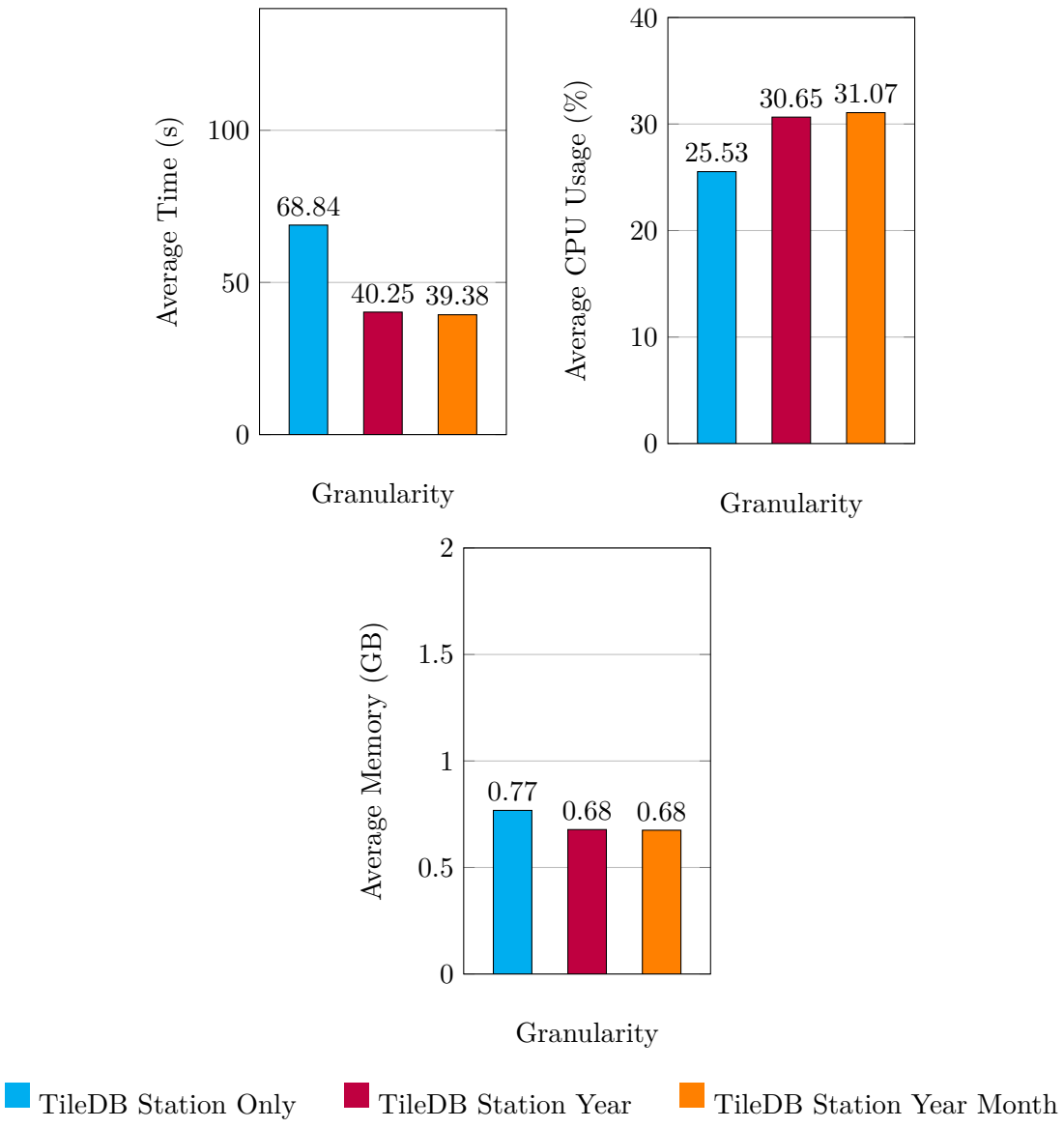


Figure 6.1: Comparison of Performance Metrics Across Granularities

and CAS100ATA_R_20242470000_01D_30S_MO.rnx occupy 77MB and 76MB, respectively, whereas MAS100ESP_R_20130950000_01D_30S_MO.rnx and MAS100ESP_R_20143290000_01D_30S_MO.rnx are significantly smaller at 22MB. Examining the tables, there is a clear increase in query execution time and total memory usage from the second pair to the first across all three granularities. This reinforces the hypothesis that larger data volumes also count to contribute to increased resource consumption.

The analysis of queries spanning the entire month of January 2019 further supports these observations, as shown in Figure 6.3. Here, the number of fragments remains constant within each granularity, but increases as the number of TileDBs decreases. Once again, the same metric correlations hold: increased fragments result in higher query times and memory usage, while reducing CPU utilization.

More intriguing, however, are the queries spanning two days from different months and years, as illustrated in Figure 6.4. These queries introduce an additional level of complexity since

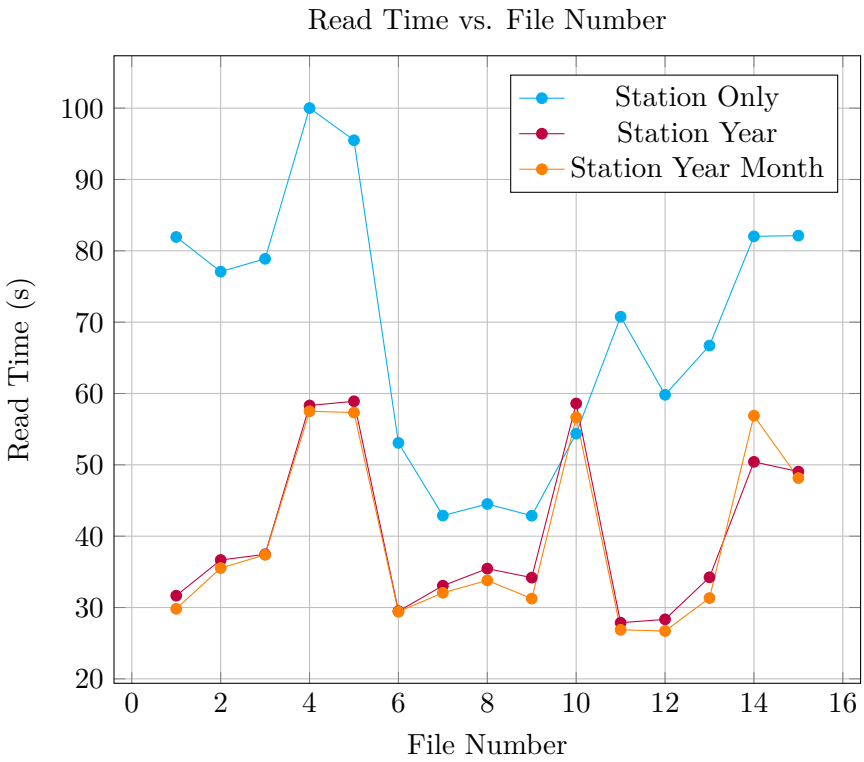


Figure 6.2: Variance of the execution time metrics across granularities.

they must access multiple TileDBs at different granularity levels. For the station and year or station, year, and month configurations, these queries require access to at least two TileDBs. In contrast, the station-only configuration still relies on a single TileDB. As a result, the differences in query metrics across the three granularities diminish significantly.
Although the general trend–where time and memory increase while CPU usage decreases as fragment count rises–still holds, the magnitude of these differences is noticeably smaller in cross-period queries. A particularly interesting case is the `TLSE00FRA` station query spanning 2023-12-31 to 2024-01-02. Here, the execution time for the station-only configuration is lower than that of the other two granularities, even though memory consumption remains higher. This suggests that in specific scenarios, the overhead of accessing multiple TileDBs outweighs the fragmentation-related slowdown seen in the station-only approach. My hypotheses for this are:

- Indexing Overhead: The need to read data across multiple TileDBs might introduce extra computational overhead;
- Merging Effects: When accessing two separate TileDBs, additional processing is required to merge results from different periods.

Another interesting observation is that all metrics appear to increase as the volume of data read grows. This trend is evident for the time and memory metrics, but it is less intuitive for the CPU metric, which actually decreased as the number of fragments increased. This counterintuitive result suggests that fragmentation may have a unique impact on CPU usage, potentially optimizing processing efficiency under certain conditions.

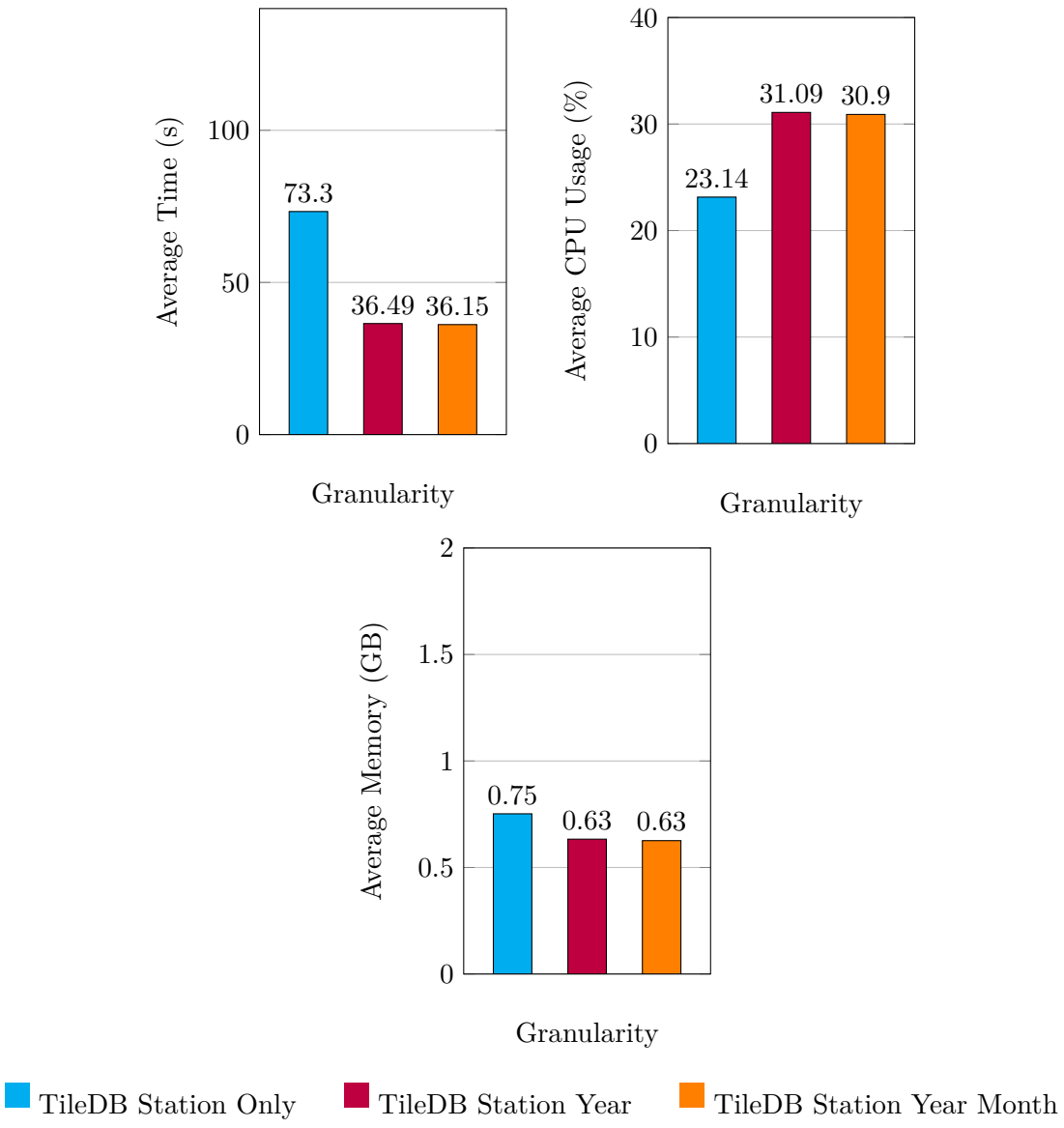


Figure 6.3: Comparison of Performance Metrics Across Granularities Spanning a Whole Month.

6.8 Conclusion

In conclusion, the results of the tests highlight the complex interplay between TileDB fragmentation, query execution time, CPU usage, and memory consumption. The key takeaway is that increasing the number of TileDBs generally results in higher memory consumption and longer query times, probably due to the increased I/O overhead. This suggests that the system can be further optimized, because the current metric results are not acceptable for a real use system. Query times of 30s+ seconds for a single day is too long.

One notable observation is the substantial increase in query times and memory usage when the number of TileDBs is reduced, especially when working with a larger number of fragments in the station-only configuration. The high volume of fragments likely contributes to this performance degradation. Moreover, the current TileDB consolidation feature, while still under testing, does not seem to efficiently handle the number of fragments present in the

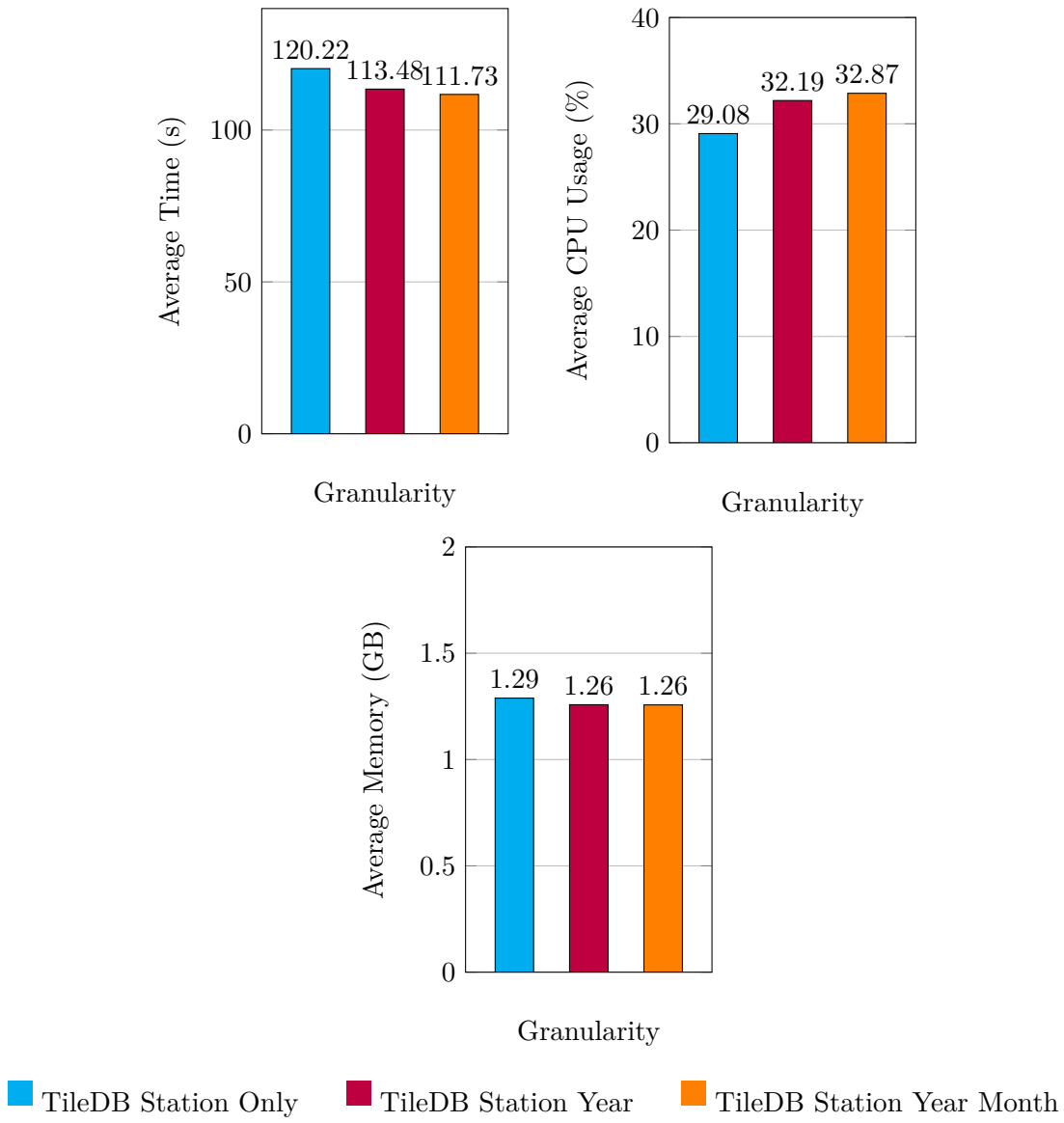


Figure 6.4: Comparison of Performance Metrics Across Granularities Spanning Two Days.

data. Future work could explore the consolidation feature in more depth, particularly with a focus on optimizing the merging of fragments to improve both query performance and memory usage.

Given that the current consolidation feature is not yet fully optimized, the prolonged query times for certain configurations can be attributed to these inefficiencies. As TileDB continues to refine its consolidation methods, the hope is that these issues will be addressed, leading to more efficient performance with fewer TileDBs.

It is important to note that the results are not conclusive, as other factors could influence performance; however, I conclude that the station-year granularity is the most promising granularity, because its performance is akin to the station-year-month granularity with substantially less TileDBs.

Chapter 7

Conclusion

This dissertation presents a comprehensive study of a unified system designed to facilitate the management, conversion, and storage of GNSS RINEX observation data, particularly focusing on the integration of various proprietary formats into a common framework. The two main contributions were:

- The `raw2rin` application, which includes a converter application, which efficiently converts RAW GNSS data from multiple vendors to the RINEX format, which, when combined with a Web API and Web/Command-line interface (CLI) clients, handles different GNSS receiver data formats, allows the changing of conversion parameters, and perform metadata corrections;
- The `tiledb_manager` application, which manages TileDBs, allowing the import of GNSS observation data between, enabling users to query and process data back into RINEX by simply issuing a few commands.

The integration of TileDB for storage was tested with various granularities of file system storage, by dividing the TileDB archive into several TileDBs according to i) station, ii) station, and year, and iii) station, year, and month. The tests revealed that although TileDB's performance is generally favorable in terms of read and write operations, certain trade-offs may exist between granularity levels; the tests indicate a correlation between both increased data size and number of fragments and higher time and memory usage, with a decrease in CPU usage.

One prominent area of future research is the consolidation of fragments within TileDB. Currently, the system does not consolidate fragments, and as a result, the number of fragments has grown to a point where consolidating them all would introduce significant overhead due to the current limitations of the consolidation feature and the hardware used. This leads to suboptimal read times, especially as the dataset grows, however, exploring methods to consolidate fragments dynamically–perhaps on a per-query basis or gradually over time–could lead to substantial performance improvements in read operations.

More testing with different granularities (two years, weekly, etc.), as well as querying more than two days could also provide valuable insights as to how the performance of TileDB deteriorates overtime.

In addition, future work could extend the system's capabilities by further enhancing error detection in RINEX data, and improving support for additional proprietary formats.

Bibliography

[ACM] ACM. [Online] https://dl.acm.org/ccs. Last accessed at January 28, 2024. 1

[Age] European Space Agency. Galileo: A constellation of navigation satellites. Accessed: January 22, 2024. Available at: https://www.esa.int/Applications/Satellite_navigation/Galileo/Galileo_a_constellation_of_navigation_satellites. 5

[Ash20] Ben Ashman. An introduction to global navigation satellite systems. Technical report, NASA Goddard Space Flight Center, 2020. Available at: https://ntrs.nasa.gov/citations/20200000265. 5

[BCR] node.bcrypt.js. [Online] https://www.npmjs.com/package/bcrypt. Last accessed at November 19, 2024. 48

[Ber23] Henry Berglund. Tiledb status at earthscope. 2023. xvii, 13, 29, 30

[BLP] Modular applications with blueprints. [Online] https://flask.palletsprojects.com/en/stable/blueprints/. Last accessed at November 4, 2024. 48

[BP17] Jacob Bolewski and Stavros Papadopoulos. Managing massive multi-dimensional array data with tiledb:—invited demo paper. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 3175–3176. IEEE, 2017. 14

[Braa] Markus Bradke. GNSS TileDB. 31, 59, 61

[Brab] Markus Bradke. PYNEX. [Online] https://git.gfz-potsdam.de/gnss/pynex. Last accessed at January 27, 2024. 31, 57

[Cen] European GNSS Service Centre. Galileo system. Accessed: January 22, 2024. Available at: https://www.gsc-europa.eu/galileo/system. 5

[CNB] High Precision GNSS Solution. [Online] https://www.comnavtech.com/blogs_details/5.html. Last accessed at January 10, 2024. 8

[COMa] Compression. [Online] https://tiledb-inc-tiledb.readthedocs-hosted.com/en/1.6.3/tutorials/compression.html. Last accessed at January 24, 2024. 20

[COMb] High Precision GNSS Solution. [Online] https://www.comnavtech.com/. Last accessed at January 10, 2024. 8

[CRU] Rinex file header/data editing. [Online] https://gnss.git-pages.gfz-potsdam.de/gfzrnx/tasks/tasks_head_edit/#show-config-file-interpretation-show_crux. Last accessed at November 10, 2024. 52

[CSR] Cross site request forgery. [Online] `https://www.blackduck.com/glossary/what-is-csrf.html`. Last accessed at November 19, 2024. 48

[DAT] Trimble. [Online] `https://support.sbg-systems.com/sc/qd/latest/reference-manual/input-formats/trimble`. Last accessed at January 10, 2024. 8

[ENC] Encryption. [Online] `https://tiledb-inc-tiledb.readthedocs-hosted.com/en/1.6.3/tutorials/encryption.html`. Last accessed at January 24, 2024. 20

[err] GNSS Positioning Techniques. [Online] `https://www.tallysman.com/gnss-positioning-techniques/`. Last accessed at January 22, 2022. 7

[FHK+11] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. An overview of the hdf5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 workshop on array databases*, pages 36–47, 2011. 27

[FIL] Filters. [Online] `https://tiledb-inc-tiledb.readthedocs-hosted.com/en/1.6.3/tutorials/filters.html`. Last accessed at January 24, 2024. 20

[FLS] Foreword - flask documentation. [Online] `https://web.archive.org/web/20171117015927/http://flask.pocoo.org/docs/0.10/foreword`. Last accessed at November 1, 2024. 47

[GCRS17] Vishakha Gupta-Cledat, Luis Remis, and Christina R Strong. Addressing the dark side of vision research: Storage. In *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*, 2017. 14

[GE07a] Werner Gurtner and Lou Estey. Rinex: The receiver independent exchange format version 2.11. 2007. 9

[GE07b] Werner Gurtner and Lou Estey. Rinex-the receiver independent exchange format-version 3.00. *Astronomical Institute, University of Bern and UNAVCO, Bolulder, Colorado.*, 2007. 9

[GFZ] GFZ Helmholtz-Zentrum Potsdam. [Online] `https://www.gfz-potsdam.de/`. Last accessed at January 14, 2024. 31

[GHH+09] Thomas L Gaussiran, Eric Hagen, R Benjamin Harris, Chris Kieschnick, Jon C Little, Richard G Mach, DC Muton, Scot L Nelsen, Colin P Petersen, David L Rainwater, et al. The gpstk: Glonass, rinex version 3.00 and more. In *Proceedings of the 22nd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2009)*, pages 1943–1954, 2009. 9

[GIS] GISGeography. How GPS Receivers Work – Trilateration vs Triangulation. [Online] `https://gisgeography.com/trilateration-triangulation-gps/`. Last accessed at January 25, 2024. xvii, 8

[Gov] U.S. Government. Space segment. Accessed: January 22, 2024. Available at: https://www.gps.gov/systems/gps/space/. 5

[Gov21] U.S. Government. Other global navigation satellite systems (gnss), 2021. Accessed: January 9, 2024. Available at: https://www.gps.gov/systems/gnss/. URL: https://www.gps.gov/systems/gnss/. 5

[Gov24] U.S. Government. Satellite navigation - gps - how it works, 2024. Accessed: January 22, 2024. Available at: https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/gps/howitworks. 5

[GSI] Leica geosystem file format - gsi file. [Online] https://jaison.info/writings/leica-geosystem-file-format-gsi-file.html. Last accessed at January 28, 2024. 8

[GSI03] Gsi online for leica tps and dna. In *GSI ONLINE for Leica TPS and DNA*, page 49. Leica Geosystems, 2003. URL: https://www.engineeringsurveyor.com/software/tps-dna-online_v114.pdf. 8

[Hat08] Yuki Hatanaka. A compression format and tools for gnss observation data. 2008. 13

[IAC] GLONASS IAC. Composition and status of the glonass orbital constellation. Accessed: January 22, 2024. Available at: https://www.glonass-iac.ru/glonass/sostavOG/. 5

[Inc15] NovAtel Inc. *An Introduction to GNSS*. ISBN: 978-0-9813754-0-3. NovAtel Inc., 2015. URL: https://en.calameo.com/read/00191579602f9b13b088e?authid=9leJ1niQkK75. xvii, 6, 7

[Jan23] Volker Janssen. Understanding the rinex format for gnss data transfer and storage. *Coordinates*, page 21, 2023. 9

[KH17] Elliott D Kaplan and Christopher Hegarty. *Understanding GPS/GNSS: Principles and Applications*. Artech House, 2017. 5

[L+15] Qian Long et al. *Parallel load and query processing in a distributed array database*. PhD thesis, Massachusetts Institute of Technology, 2015. 14

[LAN] RINEX data. [Online] https://www.lantmateriet.se/en/geodata/gps-geodesi-och-swepos/swepos/swepos-services/post-processing/rinex-data/. Last accessed at January 21, 2024. 10

[LEI] Embrace Smart Digital Heritage. [Online] https://leica-geosystems.com/. Last accessed at January 10, 2024. 8

[Li17] Hao Li. Big data technology accelerate genomics precision medicine. *arXiv preprint arXiv:1701.09045*, 2017. 14

[Mad24] MadPenguin. How to calculate cpu utilization?, 2024. Accessed: 2025-01-21. URL: https://www.madpenguin.org/how-to-calculate-cpu-utilization/. 69

[MK23] Josh Moore and Susanne Kunis. Zarr: A cloud-optimized storage for interactive access of large arrays. In *Proceedings of the Conference on Research Data Infrastructure*, volume 1, 2023. 29

[MO09] Gerald J. K. Moernaut and Daniel Orban. An introduction to bandwidth, gain pattern, polarization, and all that. In *GNNS Antennas*, Innovation | Antenna Technology, pages 1–2. GPS World, 2009. 6

[NDF+23] Yiyu Ni, Marine A. Denolle, Rob Fatland, Naomi Alterman, Bradley P. Lipovsky, and Friedrich Knuth. An Object Storage for Distributed Acoustic Sensing. *Seismological Research Letters*, 10 2023. arXiv: https://pubs.geoscienceworld.org/ssa/srl/article-pdf/doi/10.1785/0220230172/5987895/srl-2023172.1.pdf, doi:10.1785/0220230172. 28

[Nova] NovAtel. Beidou navigation satellite system (china). Accessed: January 22, 2024. Available at: https://novatel.com/an-introduction-to-gnss/gnss-constellations/beidou. 5

[Novb] NovAtel. Qzss (quasi-zenith satellite system), japan. Accessed: January 22, 2024. Available at: https://novatel.com/an-introduction-to-gnss/gnss-constellations/qzss. 5

[Ora10] Oracle. resident set size (rss), 2010. Accessed: 2025-01-22. URL: https://docs.oracle.com/cd/E29584_01/webhelp/endeca_glossary/src/gend_resident_set_size_dgraph.html. 69

[Orga] Indian Space Research Organisation. Irnss - indian regional navigation satellite system. Accessed: January 25, 2024. Available at: https://www.ursc.gov.in/navigation/irnss.jsp. 5

[Orgb] Indian Space Research Organisation. Satellite navigation services. Accessed: January 25, 2024. Available at: https://www.isro.gov.in/SatelliteNavigationServices.html. 5

[PAR] Parallelism. [Online] https://tiledb-inc-tiledb.readthedocs-hosted.com/en/1.6.3/tutorials/parallelism.html. Last accessed at January 26, 2024. 22

[PDMM16] Stavros Papadopoulos, Kushal Datta, Samuel Madden, and Timothy Mattson. The tiledb array data storage manager. *Proceedings of the VLDB Endowment*, 10(4):349–360, 2016. 12, 13, 14, 28

[Pes15] António Pestana. Reading rinex 2.11 observation data files. *Tech. Rep.*, 2015. 11, 13

[PHR+23] Vincenzo Pesce, Pablo Hermosin, Aureliano Rivolta, Shyam Bhaskaran, Stefano Silvestrini, and Andrea Colagrossi. Navigation. In *Modern Spacecraft Guidance, Navigation, and Control*, pages 441–542. Elsevier, 2023. 7

[POL+08] Mark G Petovello, Cillian O'Driscoll, Gérard Lachapelle, Daniele Borio, and Hasan Murtaza. Architecture and benefits of an advanced gnss software receiver. *Journal of Global Positioning Systems*, 7(2):156–168, 2008. 6

[RIN] RINEX data - daily files. [Online] `https://www.lantmateriet.se/en/geodata/gps-geodesi-och-swepos/swepos/swepos-services/post-processing/rinex-data---daily-files/`. Last accessed at January 15, 2024. 13

[Rom21] Ignacio Romero. The receiver independent exchange format version 4.00. In *The Receiver Independent Exchange Format Version*. IGS/RTCM RINEX WG Chair ESA/ESOC/Navigation Support Office, 2021. 7, 8, 11, 13

[Rut] Rob Rutkowski. What's the differences between the 5 gnss constellations? Accessed: January 22, 2024. Available at: `https://blog.bliley.com/the-differences-between-the-5-gnss-satellite-network-constellations`. 5

[SBF] What is SBF and where can I find more information about it? [Online] `https://customersupport.septentrio.com/s/article/What-is-SBF-and-where-can-I-find-more-information-about-it`. Last accessed at January 10, 2024. 8

[SEPa] GNSS positioning you can count on. [Online] `https://www.septentrio.com/en`. Last accessed at January 10, 2024. 8

[SEPb] GPS / GNSS receivers. [Online] `https://www.septentrio.com/en/products/gps/gnss-receivers`. Last accessed at January 10, 2024. 6

[SOU] South - Target your success. [Online] `https://www.southinstrument.com/`. Last accessed at January 10, 2024. 8

[STH] South target your success. [Online] `https://www.southinstrument.com/product/details/pro_tid/3/id/21.html`. Last accessed at January 28, 2024. 8

[sur15] surveythemark. Leica gps file extensions, 2015. Accessed: January 31, 2025. Available at: `https://rpls.com/forums/gnss-geodesy/leica-gps-file-extensions/`. 8

[Tec] VectorNav Technologies. 1.2 global navigation satellite system (gnss). Accessed: January 23, 2024. Available at: `https://www.vectornav.com/resources/inertial-navigation-primer/theory-of-operation/theory-gnss`. 5

[Tec24] TechyGuid. How to calculate cpu usage – optimize the performance today!, 2024. Accessed: 2025-01-21. URL: https://www.techyguid.com/how-to-calculate-cpu-usage/. 69

[TEX] RINEX: The GNSS Data Exchange Format. [Online] https://mettatec.com/rinex-the-gnss-data-exchange-format/. Last accessed at January 15, 2024. 13

[Tho21] Will Thornton. Unsure of the difference between gnss and gps? keep reading for a brief history and introduction, 2021. Accessed: January 8, 2024. Available at: https://www.spirent.com/blogs/what-is-the-difference-between-gnss-and-gps. 5

[Til24] Inc. TileDB. Fragments and consolidation, 2024. Accessed: January 31, 2025. Available at: https://tiledb-inc-tiledb.readthedocs-hosted.com/en/1.6.3/tutorials/fragments-consolidation.html. 70

[Tra22] Bruce Tran. Rinex 3 provide higher precision of cors positions. URL: https://geodesy.noaa.gov/web/science_edu/presentations_library/files/webinar_rinex3.pdf, 07 2022. 13

[TRI] A new approach to work. [Online] https://www.trimble.com/en. Last accessed at January 10, 2024. 8

[UNAa] MEASURING OUR CHANGING EARTH. [Online] https://www.unavco.org/. Last accessed at January 14, 2024. 29

[UNAb] Storage of gnss data in the cloud. [Online] https://nextcloud.gfz-potsdam.de/s/WSzPT66sscmiNMW. Last accessed at January 28, 2024. 30

[UNAc] UNAVCO Research Data. [Online] https://www.un-spider.org/unavco-research-data. Last accessed at January 27, 2024. 29

[Was] Leah A. Wasser. Hierarchical data formats - what is hdf5? [Online] https://www.neonscience.org/resources/learning-hub/tutorials/about-hdf5. Last accessed at January 28, 2024. xvii, 28

[WFL] Why should you use flask framework for web development? [Online] https://medium.com/@lauren-fox/why-should-you-use-flask-framework-for-web-development-f5a7233e17a6. Last accessed at November 4, 2024. 48

[WTF] Wtforms. [Online] https://wtforms.readthedocs.io/en/3.2.x/. Last accessed at November 4, 2024. 48

Appendix A

Appendix

A.1 Full Size Web Client Images

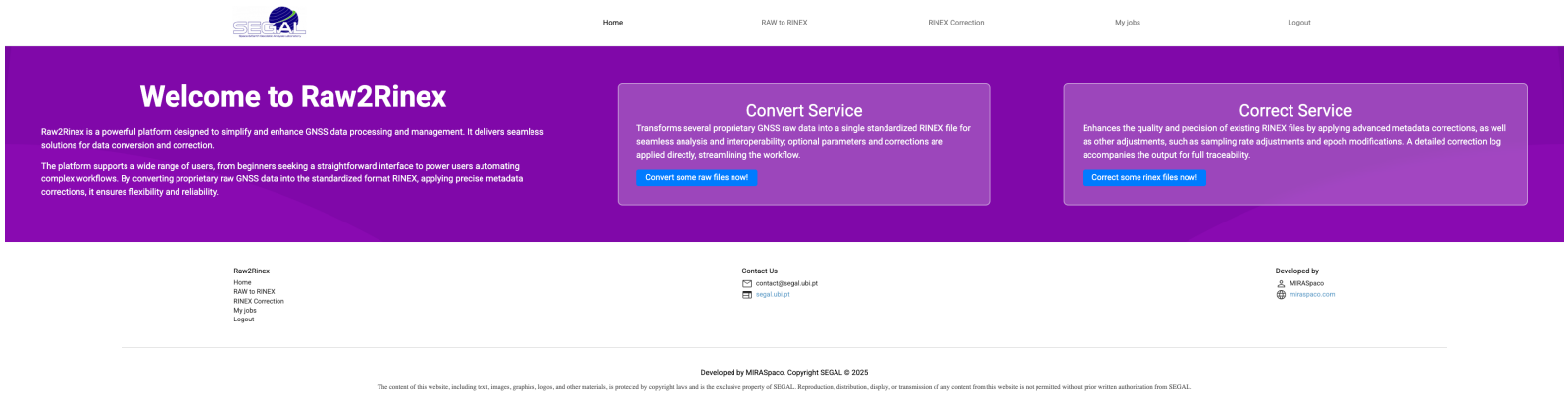


Figure A.1: Full-size home page on the web client.

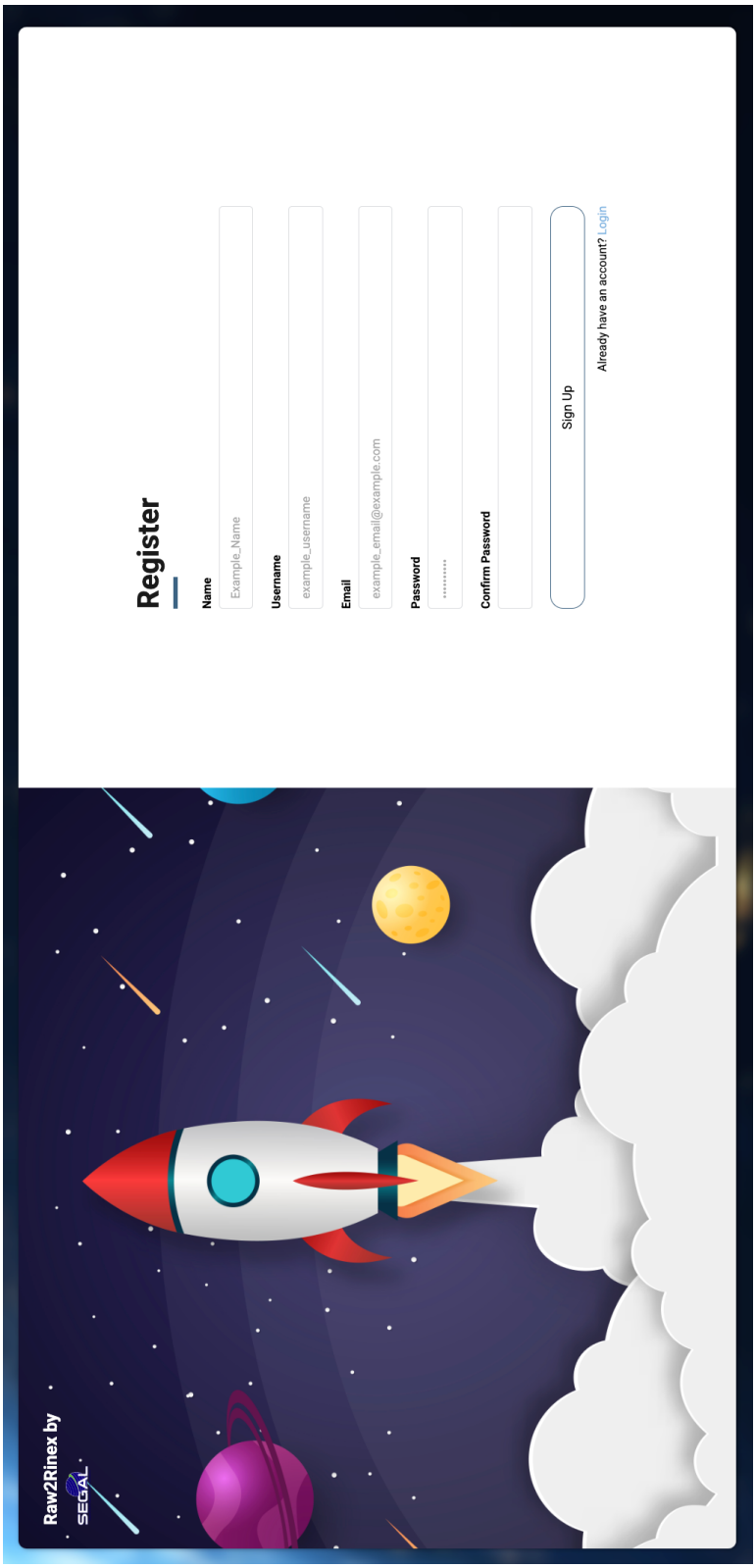


Figure A.2: Full-size register page on the web client.

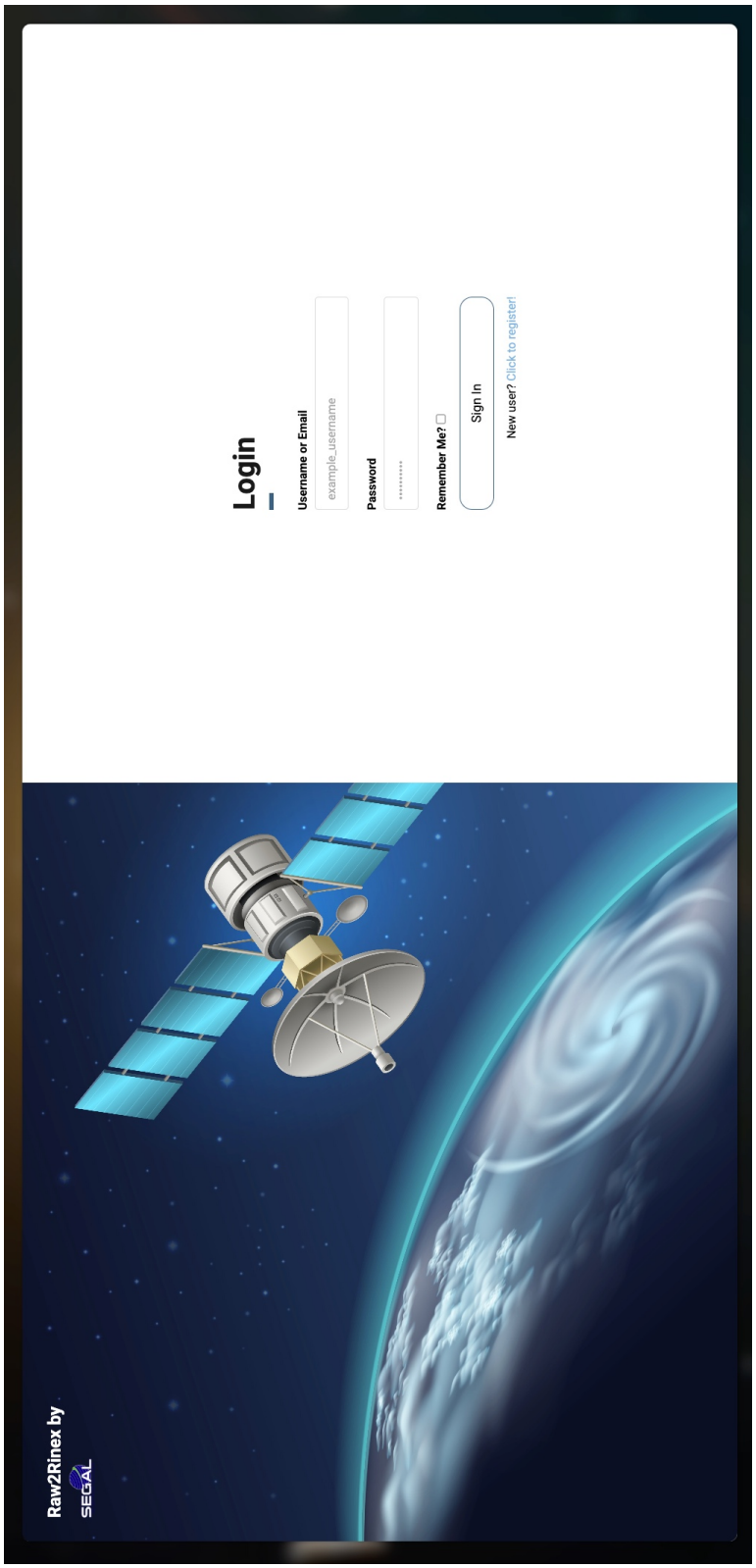


Figure A.3: Full-size login page on the web client.

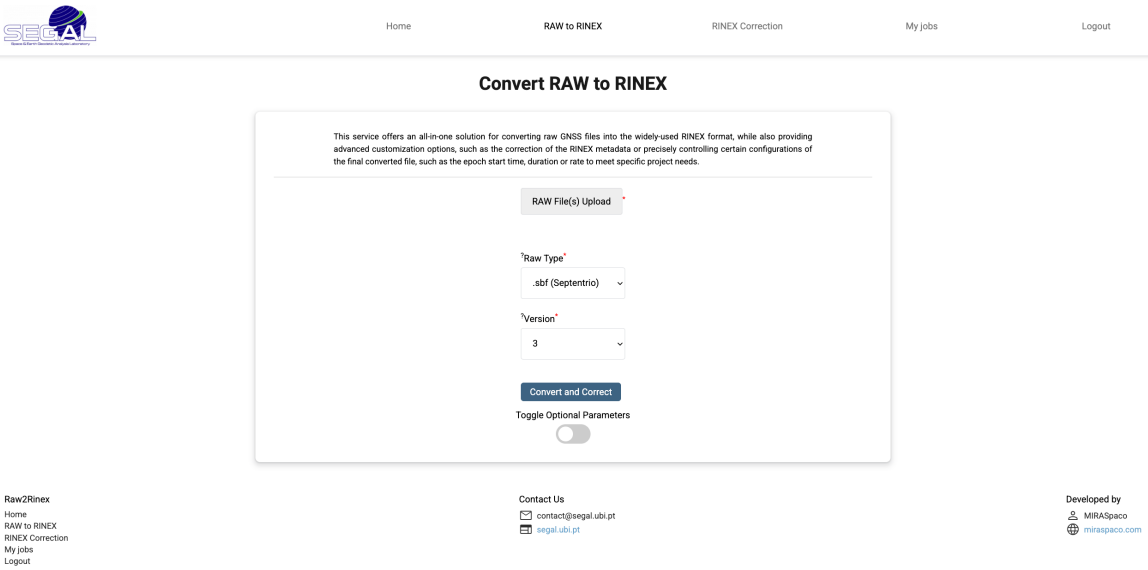


Figure A.4: Full-size convert and correct page on the web client.

Figure A.5: Full-size convert and correct page with optional fields on the web client.

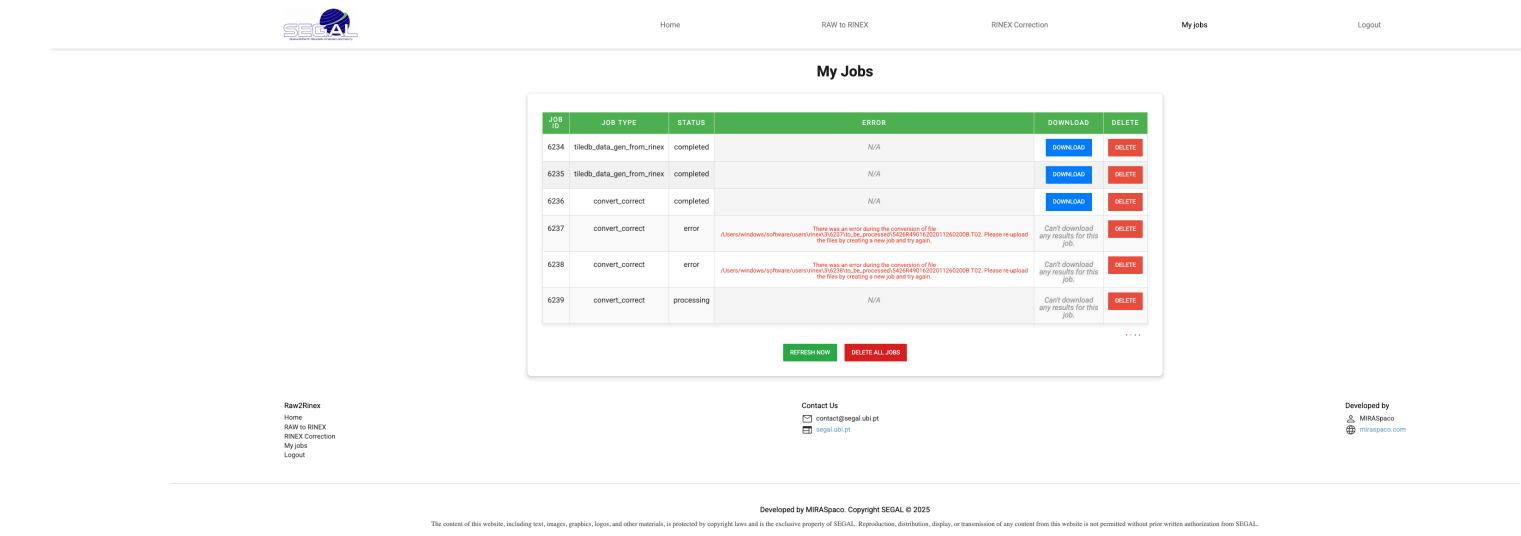


Figure A.6: Full-size jobs page on the web client.

A.2 IGS-Type Log File

Listing A.1: IGS-Type log file template

```
 XXXX Site Information Form (site log)
      International GNSS Service
      See Instructions at:
        https://files.igs.org/pub/station/general/sitelog_instr.txt

0.   Form

      Prepared by (full name)  :
      Date Prepared            : (CCYY-MM-DD)
      Report Type              : (NEW/UPDATE)
      If Update:
       Previous Site Log       : (ssss_ccyymmdd.log)
       Modified/Added Sections : (n.n,n.n,...)


1.   Site Identification of the GNSS Monument

      Site Name                :
      Four Character ID        : (A4)
      Monument Inscription     :
      IERS DOMES Number        : (A9)
      CDP Number               : (A4)
      Monument Description     : (PILLAR/BRASS PLATE/STEEL MAST/etc)
        Height of the Monument : (m)
        Monument Foundation    : (STEEL RODS, CONCRETE BLOCK, ROOF, etc)
        Foundation Depth       : (m)
      Marker Description       : (CHISELLED CROSS/DIVOT/BRASS NAIL/etc)
      Date Installed           : (CCYY-MM-DDThh:mmZ)
      Geologic Characteristic  : (BEDROCK/CLAY/CONGLOMERATE/GRAVEL/SAND/etc)
        Bedrock Type           : (IGNEOUS/METAMORPHIC/SEDIMENTARY)
        Bedrock Condition      : (FRESH/JOINTED/WEATHERED)
        Fracture Spacing       : (1-10 cm/11-50 cm/51-200 cm/over 200 cm)
        Fault zones nearby     : (YES/NO/Name of the zone)
          Distance/activity    : (multiple lines)
      Additional Information   : (multiple lines)


2.   Site Location Information

      City or Town             :
      State or Province        :
      Country                  :
      Tectonic Plate           :
      Approximate Position (ITRF)
        X coordinate (m)       :
        Y coordinate (m)       :
        Z coordinate (m)       :
        Latitude (N is +)      : (+/-DDMMSS.SS)
        Longitude (E is +)     : (+/-DDDMMSS.SS)
        Elevation (m,ellips.)  : (F7.1)
      Additional Information   : (multiple lines)


3.   GNSS Receiver Information
```

```
55
56  3.1  Receiver Type            : (A20, from rcvr_ant.tab; see instructions)
57       Satellite System         : (GPS+GLO+GAL+BDS+QZSS+SBAS)
58       Serial Number            : (A20, but note the first A5 is used in SINEX)
59       Firmware Version         : (A11)
60       Elevation Cutoff Setting : (deg)
61       Date Installed           : (CCYY-MM-DDThh:mmZ)
62       Date Removed             : (CCYY-MM-DDThh:mmZ)
63       Temperature Stabiliz.    : (none or tolerance in degrees C)
64       Additional Information   : (multiple lines)
65
66  3.x  Receiver Type            : (A20, from rcvr_ant.tab; see instructions)
67       Satellite System         : (GPS+GLO+GAL+BDS+QZSS+SBAS)
68       Serial Number            : (A20, but note the first A5 is used in SINEX)
69       Firmware Version         : (A11)
70       Elevation Cutoff Setting : (deg)
71       Date Installed           : (CCYY-MM-DDThh:mmZ)
72       Date Removed             : (CCYY-MM-DDThh:mmZ)
73       Temperature Stabiliz.    : (none or tolerance in degrees C)
74       Additional Information   : (multiple lines)
75
76
77  4.   GNSS Antenna Information
78
79  4.1  Antenna Type             : (A20, from rcvr_ant.tab; see instructions)
80       Serial Number            : (A*, but note the first A5 is used in SINEX)
81       Antenna Reference Point  : (BPA/BCR/XXX from "antenna.gra"; see instr.)
82       Marker->ARP Up Ecc. (m)  : (F8.4)
83       Marker->ARP North Ecc(m) : (F8.4)
84       Marker->ARP East Ecc(m)  : (F8.4)
85       Alignment from True N    : (deg; + is clockwise/east)
86       Antenna Radome Type      : (A4 from rcvr_ant.tab; see instructions)
87       Radome Serial Number     :
88       Antenna Cable Type       : (vendor & type number)
89       Antenna Cable Length     : (m)
90       Date Installed           : (CCYY-MM-DDThh:mmZ)
91       Date Removed             : (CCYY-MM-DDThh:mmZ)
92       Additional Information   : (multiple lines)
93
94  4.x  Antenna Type             : (A20, from rcvr_ant.tab; see instructions)
95       Serial Number            : (A*, but note the first A5 is used in SINEX)
96       Antenna Reference Point  : (BPA/BCR/XXX from "antenna.gra"; see instr.)
97       Marker->ARP Up Ecc. (m)  : (F8.4)
98       Marker->ARP North Ecc(m) : (F8.4)
99       Marker->ARP East Ecc(m)  : (F8.4)
100      Alignment from True N    : (deg; + is clockwise/east)
101      Antenna Radome Type      : (A4 from rcvr_ant.tab; see instructions)
102      Radome Serial Number     :
103      Antenna Cable Type       : (vendor & type number)
104      Antenna Cable Length     : (m)
105      Date Installed           : (CCYY-MM-DDThh:mmZ)
106      Date Removed             : (CCYY-MM-DDThh:mmZ)
107      Additional Information   : (multiple lines)
108
109 5.   Surveyed Local Ties
110
111 5.x  Tied Marker Name         :
112      Tied Marker Usage        : (SLR/VLBI/LOCAL CONTROL/FOOTPRINT/etc)
113      Tied Marker CDP Number   : (A4)
114      Tied Marker DOMES Number : (A9)
115      Differential Components from GNSS Marker to the tied monument (ITRS)
```

```
       dx (m)                 : (m)
       dy (m)                 : (m)
       dz (m)                 : (m)
     Accuracy (mm)            : (mm)
     Survey method            : (GPS CAMPAIGN/TRILATERATION/TRIANGULATION/etc)
     Date Measured            : (CCYY-MM-DDThh:mmZ)
     Additional Information   : (multiple lines)


6.   Frequency Standard

6.1  Standard Type            : (INTERNAL or EXTERNAL H-MASER/CESIUM/etc)
       Input Frequency        : (if external)
       Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
       Notes                  : (multiple lines)

6.x  Standard Type            : (INTERNAL or EXTERNAL H-MASER/CESIUM/etc)
       Input Frequency        : (if external)
       Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
       Notes                  : (multiple lines)


7.   Collocation Information

7.1  Instrumentation Type     : (GPS/GLONASS/DORIS/PRARE/SLR/VLBI/TIME/etc)
       Status                 : (PERMANENT/MOBILE)
       Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
       Notes                  : (multiple lines)

7.x  Instrumentation Type     : (GPS/GLONASS/DORIS/PRARE/SLR/VLBI/TIME/etc)
       Status                 : (PERMANENT/MOBILE)
       Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
       Notes                  : (multiple lines)


8.   Meteorological Instrumentation

8.1.1 Humidity Sensor Model   :
       Manufacturer           :
       Serial Number          :
       Data Sampling Interval : (sec)
       Accuracy (% rel h)     : (% rel h)
       Aspiration             : (UNASPIRATED/NATURAL/FAN/etc)
       Height Diff to Ant     : (m)
       Calibration date       : (CCYY-MM-DD)
       Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
       Notes                  : (multiple lines)

8.1.x Humidity Sensor Model   :
       Manufacturer           :
       Serial Number          :
       Data Sampling Interval : (sec)
       Accuracy (% rel h)     : (% rel h)
       Aspiration             : (UNASPIRATED/NATURAL/FAN/etc)
       Height Diff to Ant     : (m)
       Calibration date       : (CCYY-MM-DD)
       Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
       Notes                  : (multiple lines)

8.2.1 Pressure Sensor Model   :
       Manufacturer           :
```

```
       Serial Number          :
       Data Sampling Interval : (sec)
       Accuracy               : (hPa)
       Height Diff to Ant     : (m)
       Calibration date       : (CCYY-MM-DD)
       Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
       Notes                  : (multiple lines)

8.2.x Pressure Sensor Model   :
       Manufacturer           :
       Serial Number          :
       Data Sampling Interval : (sec)
       Accuracy               : (hPa)
       Height Diff to Ant     : (m)
       Calibration date       : (CCYY-MM-DD)
       Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
       Notes                  : (multiple lines)

8.3.1 Temp. Sensor Model      :
       Manufacturer           :
       Serial Number          :
       Data Sampling Interval : (sec)
       Accuracy               : (deg C)
       Aspiration             : (UNASPIRATED/NATURAL/FAN/etc)
       Height Diff to Ant     : (m)
       Calibration date       : (CCYY-MM-DD)
       Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
       Notes                  : (multiple lines)

8.3.x Temp. Sensor Model      :
       Manufacturer           :
       Serial Number          :
       Data Sampling Interval : (sec)
       Accuracy               : (deg C)
       Aspiration             : (UNASPIRATED/NATURAL/FAN/etc)
       Height Diff to Ant     : (m)
       Calibration date       : (CCYY-MM-DD)
       Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
       Notes                  : (multiple lines)

8.4.1 Water Vapor Radiometer  :
       Manufacturer           :
       Serial Number          :
       Distance to Antenna    : (m)
       Height Diff to Ant     : (m)
       Calibration date       : (CCYY-MM-DD)
       Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
       Notes                  : (multiple lines)

8.4.x Water Vapor Radiometer  :
       Manufacturer           :
       Serial Number          :
       Distance to Antenna    : (m)
       Height Diff to Ant     : (m)
       Calibration date       : (CCYY-MM-DD)
       Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
       Notes                  : (multiple lines)

8.5.1 Other Instrumentation   : (multiple lines)

8.5.x Other Instrumentation   : (multiple lines)
```

```
238
239
240 9.  Local Ongoing Conditions Possibly Affecting Computed Position
241
242 9.1.1 Radio Interferences     : (TV/CELL PHONE ANTENNA/RADAR/etc)
243        Observed Degradations  : (SN RATIO/DATA GAPS/etc)
244        Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
245        Additional Information : (multiple lines)
246
247 9.1.x Radio Interferences     : (TV/CELL PHONE ANTENNA/RADAR/etc)
248        Observed Degradations  : (SN RATIO/DATA GAPS/etc)
249        Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
250        Additional Information : (multiple lines)
251
252 9.2.1 Multipath Sources       : (METAL ROOF/DOME/VLBI ANTENNA/etc)
253        Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
254        Additional Information : (multiple lines)
255
256 9.2.x Multipath Sources       : (METAL ROOF/DOME/VLBI ANTENNA/etc)
257        Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
258        Additional Information : (multiple lines)
259
260 9.3.1 Signal Obstructions     : (TREES/BUILDINGS/etc)
261        Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
262        Additional Information : (multiple lines)
263
264 9.3.x Signal Obstructions     : (TREES/BUILDINGS/etc)
265        Effective Dates        : (CCYY-MM-DD/CCYY-MM-DD)
266        Additional Information : (multiple lines)
267
268 10.  Local Episodic Effects Possibly Affecting Data Quality
269
270 10.1 Date                    : (CCYY-MM-DD/CCYY-MM-DD)
271      Event                   : (TREE CLEARING/CONSTRUCTION/etc)
272
273 10.x Date                    : (CCYY-MM-DD/CCYY-MM-DD)
274      Event                   : (TREE CLEARING/CONSTRUCTION/etc)
275
276 11.   On-Site, Point of Contact Agency Information
277
278      Agency                  : (multiple lines)
279      Preferred Abbreviation  : (A10)
280      Mailing Address         : (multiple lines)
281      Primary Contact
282        Contact Name          :
283        Telephone (primary)   :
284        Telephone (secondary) :
285        Fax                   :
286        E-mail                :
287      Secondary Contact
288        Contact Name          :
289        Telephone (primary)   :
290        Telephone (secondary) :
291        Fax                   :
292        E-mail                :
293      Additional Information  : (multiple lines)
294
295
296 12.  Responsible Agency (if different from 11.)
297
298      Agency                  : (multiple lines)
```

```
     Preferred Abbreviation   : (A10)
     Mailing Address          : (multiple lines)
     Primary Contact
       Contact Name           :
       Telephone (primary)    :
       Telephone (secondary)  :
       Fax                    :
       E-mail                 :
     Secondary Contact
       Contact Name           :
       Telephone (primary)    :
       Telephone (secondary)  :
       Fax                    :
       E-mail                 :
     Additional Information   : (multiple lines)


13.  More Information

     Primary Data Center      :
     Secondary Data Center    :
     URL for More Information :
     Hardcopy on File
       Site Map               : (Y or URL)
       Site Diagram           : (Y or URL)
       Horizon Mask           : (Y or URL)
       Monument Description   : (Y or URL)
       Site Pictures          : (Y or URL)
     Additional Information   : (multiple lines)
     Antenna Graphics with Dimensions

     (insert text graphic from file antenna.gra)
```

A.3 Resulting Tables for the Different Queries

Station	**Start Epoch**	**End Epoch**	**Number of Fragments**	**Time (s)**	**Avg CPU Usage** (%)	**Total Memory (GB)**
CAS100ATA	2015-07-13	2015-07-14	11520	81.94s	20.60%	0.617GB
CAS100ATA	2018-12-27	2018-12-28	11520	77.08s	23.40%	0.755GB
CAS100ATA	2020-01-16	2020-01-17	11520	78.88s	23.50%	0.798GB
CAS100ATA	2022-03-07	2022-03-08	11520	100.01s	24.20%	1.188GB
CAS100ATA	2024-09-03	2024-09-04	11520	95.50s	23.80%	1.171GB
TLSE00FRA	2016-02-25	2016-02-26	7410	53.08s	26.40%	0.560GB
TLSE00FRA	2017-10-26	2017-10-27	7410	42.89s	29.00%	0.613GB
TLSE00FRA	2019-08-21	2019-08-22	7410	44.51s	29.30%	0.642GB
TLSE00FRA	2021-04-21	2021-04-22	7410	42.87s	29.10%	0.612GB
TLSE00FRA	2023-12-31	2024-01-01	7410	54.37s	30.10%	0.873GB
MAS100ESP	2013-04-05	2013-04-06	10767	70.77s	23.20%	0.543GB
MAS100ESP	2014-11-25	2014-11-26	10767	59.82s	24.10%	0.547GB
MAS100ESP	2017-05-27	2017-05-28	10767	66.72s	24.40%	0.652GB
MAS100ESP	2021-03-31	2021-04-01	10767	82.03s	26.30%	0.957GB
MAS100ESP	2024-06-07	2024-06-08	10767	82.13s	25.60%	0.986GB
Total Average	-	-	9899	68.84s	25.53%	0.768GB

Table A.1: Query Results for TileDB station only

Station	**Start Epoch**	**End Epoch**	**Number of Fragments**	**Time (s)**	**Avg CPU Usage** (%)	**Total Memory (GB)**
CAS100ATA	2015-07-13	2015-07-14	993	31.66s	30.60%	0.495GB
CAS100ATA	2018-12-27	2018-12-28	1092	36.66s	30.70%	0.635GB
CAS100ATA	2020-01-16	2020-01-17	75	37.46s	31.10%	0.671GB
CAS100ATA	2022-03-07	2022-03-08	216	58.32s	31.10%	1.095GB
CAS100ATA	2024-09-03	2024-09-04	768	58.91s	30.70%	1.080GB
TLSE00FRA	2016-02-25	2016-02-26	861	29.48s	30.30%	0.486GB
TLSE00FRA	2017-10-26	2017-10-27	1095	33.06s	30.20%	0.546GB
TLSE00FRA	2019-08-21	2019-08-22	1095	35.45s	30.60%	0.584GB
TLSE00FRA	2021-04-21	2021-04-22	1086	34.19s	30.60%	0.548GB
TLSE00FRA	2023-12-31	2024-01-01	1086	58.60s	31.00%	0.847GB
MAS100ESP	2013-04-05	2013-04-06	1044	27.87s	30.30%	0.429GB
MAS100ESP	2014-11-25	2014-11-26	1089	28.34s	30.40%	0.434GB
MAS100ESP	2017-05-27	2017-05-28	1089	34.24s	30.20%	0.539GB
MAS100ESP	2021-03-31	2021-04-01	1086	50.42s	30.80%	0.875GB
MAS100ESP	2024-06-07	2024-06-08	783	49.05s	31.10%	0.900GB
Total Average	-	-	897	40.25s	30.65%	0.678GB

Table A.2: Query Results for TileDB station year

Station	**Start Epoch**	**End Epoch**	**Number of Fragments**	**Time (s)**	**Avg CPU Usage** (%)	**Total Memory (GB)**
CAS100ATA	2015-07-1	2015-07-14	42	29.82s	31.00%	0.488GB
CAS100ATA	2018-12-2	2018-12-28	93	35.53s	31.30%	0.628GB
CAS100ATA	2020-01-1	2020-01-17	75	37.39s	30.80%	0.671GB
CAS100ATA	2022-03-0	2022-03-08	54	57.51s	31.00%	1.094GB
CAS100ATA	2024-09-0	2024-09-04	51	57.33s	31.20%	1.077GB
TLSE00FRA	2016-02-2	2016-02-26	81	29.41s	31.00%	0.480GB
TLSE00FRA	2017-10-2	2017-10-27	93	32.07s	30.80%	0.538GB
TLSE00FRA	2019-08-2	2019-08-22	93	33.80s	30.90%	0.577GB
TLSE00FRA	2021-04-2	2021-04-22	90	31.25s	31.40%	0.541GB
TLSE00FRA	2023-12-3	2024-01-01	93	56.64s	31.40%	0.846GB
MAS100ESP	2013-04-0	2013-04-06	87	26.88s	31.10%	0.421GB
MAS100ESP	2014-11-2	2014-11-26	90	26.71s	30.90%	0.426GB
MAS100ESP	2017-05-2	2017-05-28	93	31.33s	30.80%	0.532GB
MAS100ESP	2021-03-3	2021-04-01	90	56.89s	31.30%	0.906GB
MAS100ESP	2024-06-0	2024-06-08	90	48.15s	31.10%	0.897GB
Total Average	-	-	81	39.38s	31.07%	0.675GB

Table A.3: Query Results for TileDB station year month

Station	**Start Epoch**	**End Epoch**	**Time (s)**	**Avg CPU Usage** (%)	**Total Memory (GB)**
CAS100ATA	2019-01-01	2019-01-02	141.21s	14.00%	0.746GB
CAS100ATA	2019-01-02	2019-01-03	69.70s	23.40%	0.746GB
CAS100ATA	2019-01-03	2019-01-04	71.88s	23.40%	0.744GB
CAS100ATA	2019-01-04	2019-01-05	71.34s	23.40%	0.752GB
CAS100ATA	2019-01-05	2019-01-06	70.01s	23.80%	0.751GB
CAS100ATA	2019-01-06	2019-01-07	71.05s	23.80%	0.751GB
CAS100ATA	2019-01-07	2019-01-08	71.32s	24.00%	0.751GB
CAS100ATA	2019-01-08	2019-01-09	73.04s	23.60%	0.753GB
CAS100ATA	2019-01-09	2019-01-10	70.90s	23.80%	0.755GB
CAS100ATA	2019-01-10	2019-01-11	71.87s	23.20%	0.752GB
CAS100ATA	2019-01-11	2019-01-12	71.93s	23.20%	0.749GB
CAS100ATA	2019-01-12	2019-01-13	71.43s	22.80%	0.755GB
CAS100ATA	2019-01-13	2019-01-14	71.18s	23.40%	0.750GB
CAS100ATA	2019-01-14	2019-01-15	72.47s	22.80%	0.751GB
CAS100ATA	2019-01-15	2019-01-16	70.45s	23.20%	0.746GB
CAS100ATA	2019-01-16	2019-01-17	70.60s	23.40%	0.749GB
CAS100ATA	2019-01-17	2019-01-18	70.05s	23.60%	0.751GB
CAS100ATA	2019-01-18	2019-01-19	69.56s	23.60%	0.746GB
CAS100ATA	2019-01-19	2019-01-20	71.27s	23.60%	0.747GB
CAS100ATA	2019-01-20	2019-01-21	70.43s	23.60%	0.747GB
CAS100ATA	2019-01-21	2019-01-22	69.93s	23.60%	0.747GB
CAS100ATA	2019-01-22	2019-01-23	69.72s	23.60%	0.736GB
CAS100ATA	2019-01-23	2019-01-24	70.18s	24.00%	0.749GB
CAS100ATA	2019-01-24	2019-01-25	71.44s	23.40%	0.758GB
CAS100ATA	2019-01-25	2019-01-26	71.00s	23.20%	0.764GB
CAS100ATA	2019-01-26	2019-01-27	72.05s	22.80%	0.765GB
CAS100ATA	2019-01-27	2019-01-28	72.95s	23.20%	0.758GB
CAS100ATA	2019-01-28	2019-01-29	71.13s	23.60%	0.755GB
CAS100ATA	2019-01-29	2019-01-30	69.81s	23.20%	0.759GB
CAS100ATA	2019-01-30	2019-01-31	70.63s	23.60%	0.768GB
CAS100ATA	2019-01-31	2019-02-01	71.84s	23.60%	0.765GB
Total Average	-	-	73.30s	23.14%	0.752GB

Table A.4: Query Results for a whole month for TileDB station only.

Station	**Start Epoch**	**End Epoch**	**Time (s)**	**Avg CPU Usage** (%)	**Total Memory (GB)**
CAS100ATA	2019-01-01	2019-01-02	46.80s	25.80%	0.625GB
CAS100ATA	2019-01-02	2019-01-03	36.24s	31.40%	0.627GB
CAS100ATA	2019-01-03	2019-01-04	35.70s	31.00%	0.625GB
CAS100ATA	2019-01-04	2019-01-05	36.92s	31.00%	0.630GB
CAS100ATA	2019-01-05	2019-01-06	36.12s	31.20%	0.633GB
CAS100ATA	2019-01-06	2019-01-07	35.47s	31.20%	0.632GB
CAS100ATA	2019-01-07	2019-01-08	36.34s	31.20%	0.630GB
CAS100ATA	2019-01-08	2019-01-09	36.53s	31.40%	0.631GB
CAS100ATA	2019-01-09	2019-01-10	36.33s	31.00%	0.634GB
CAS100ATA	2019-01-10	2019-01-11	36.16s	31.20%	0.629GB
CAS100ATA	2019-01-11	2019-01-12	35.97s	31.40%	0.630GB
CAS100ATA	2019-01-12	2019-01-13	36.77s	31.40%	0.634GB
CAS100ATA	2019-01-13	2019-01-14	36.63s	31.80%	0.631GB
CAS100ATA	2019-01-14	2019-01-15	36.39s	31.40%	0.629GB
CAS100ATA	2019-01-15	2019-01-16	35.96s	31.40%	0.631GB
CAS100ATA	2019-01-16	2019-01-17	36.46s	31.00%	0.631GB
CAS100ATA	2019-01-17	2019-01-18	36.11s	31.40%	0.630GB
CAS100ATA	2019-01-18	2019-01-19	35.78s	31.60%	0.625GB
CAS100ATA	2019-01-19	2019-01-20	35.98s	31.40%	0.630GB
CAS100ATA	2019-01-20	2019-01-21	35.83s	31.40%	0.626GB
CAS100ATA	2019-01-21	2019-01-22	35.91s	31.60%	0.625GB
CAS100ATA	2019-01-22	2019-01-23	35.16s	31.20%	0.615GB
CAS100ATA	2019-01-23	2019-01-24	35.58s	31.20%	0.635GB
CAS100ATA	2019-01-24	2019-01-25	36.47s	31.00%	0.639GB
CAS100ATA	2019-01-25	2019-01-26	36.41s	31.20%	0.643GB
CAS100ATA	2019-01-26	2019-01-27	36.23s	31.20%	0.645GB
CAS100ATA	2019-01-27	2019-01-28	35.66s	31.40%	0.641GB
CAS100ATA	2019-01-28	2019-01-29	37.10s	30.80%	0.638GB
CAS100ATA	2019-01-29	2019-01-30	35.58s	31.00%	0.642GB
CAS100ATA	2019-01-30	2019-01-31	36.74s	31.40%	0.647GB
CAS100ATA	2019-01-31	2019-02-01	35.85s	31.20%	0.645GB
Total Average	-	-	36.49s	31.09%	0.633GB

Table A.5: Query Results for a whole month for TileDB station, and year.

Station	Start Epoch	End Epoch	Time (s)	Avg CPU Usage (%)	Total Memory (GB)
CAS100ATA	2019-01-01	2019-01-02	36.89s	29.80%	0.618GB
CAS100ATA	2019-01-02	2019-01-03	35.47s	30.80%	0.620GB
CAS100ATA	2019-01-03	2019-01-04	36.01s	30.60%	0.618GB
CAS100ATA	2019-01-04	2019-01-05	35.96s	30.80%	0.623GB
CAS100ATA	2019-01-05	2019-01-06	35.74s	31.00%	0.626GB
CAS100ATA	2019-01-06	2019-01-07	35.76s	31.60%	0.625GB
CAS100ATA	2019-01-07	2019-01-08	35.64s	31.40%	0.622GB
CAS100ATA	2019-01-08	2019-01-09	35.89s	30.80%	0.624GB
CAS100ATA	2019-01-09	2019-01-10	35.12s	30.80%	0.626GB
CAS100ATA	2019-01-10	2019-01-11	34.91s	31.00%	0.622GB
CAS100ATA	2019-01-11	2019-01-12	35.13s	30.80%	0.623GB
CAS100ATA	2019-01-12	2019-01-13	35.14s	30.40%	0.627GB
CAS100ATA	2019-01-13	2019-01-14	34.94s	30.40%	0.624GB
CAS100ATA	2019-01-14	2019-01-15	34.94s	30.80%	0.622GB
CAS100ATA	2019-01-15	2019-01-16	35.46s	31.00%	0.624GB
CAS100ATA	2019-01-16	2019-01-17	35.23s	30.80%	0.624GB
CAS100ATA	2019-01-17	2019-01-18	35.06s	31.20%	0.622GB
CAS100ATA	2019-01-18	2019-01-19	35.36s	31.20%	0.618GB
CAS100ATA	2019-01-19	2019-01-20	34.88s	30.80%	0.623GB
CAS100ATA	2019-01-20	2019-01-21	35.43s	31.20%	0.619GB
CAS100ATA	2019-01-21	2019-01-22	35.36s	31.40%	0.618GB
CAS100ATA	2019-01-22	2019-01-23	35.46s	30.80%	0.608GB
CAS100ATA	2019-01-23	2019-01-24	36.13s	30.80%	0.628GB
CAS100ATA	2019-01-24	2019-01-25	36.04s	31.20%	0.632GB
CAS100ATA	2019-01-25	2019-01-26	36.68s	30.60%	0.635GB
CAS100ATA	2019-01-26	2019-01-27	36.03s	31.00%	0.638GB
CAS100ATA	2019-01-27	2019-01-28	36.68s	31.00%	0.634GB
CAS100ATA	2019-01-28	2019-01-29	36.57s	31.00%	0.631GB
CAS100ATA	2019-01-29	2019-01-30	37.12s	30.80%	0.634GB
CAS100ATA	2019-01-30	2019-01-31	36.89s	31.40%	0.639GB
CAS100ATA	2019-01-31	2019-02-01	48.70s	30.60%	0.661GB
Total Average	-	-	36.15s	30.90%	0.626GB

Table A.6: Query Results for a whole month for TileDB station, year, and month.

Station	Start Epoch	End Epoch	Number of Fragments	Time (s)	Avg CPU Usage (%)	Total Memory (GB)
CAS100ATA	2017-12-31	2018-01-02	11520	111.60s	28.24%	1.098GB
TLSE00FRA	2019-12-31	2020-01-02	7410	95.25s	30.72%	1.055GB
MAS100ESP	2023-12-31	2024-01-02	10767	153.79s	28.28%	1.713GB
Total Average	-	-	9899	120.22s	29.08%	1.289GB

Table A.7: Query Results for two days for TileDB station only.

Station	Start Epoch	End Epoch	Number of Fragments	Time (s)	Avg CPU Usage (%)	Total Memory (GB)
CAS100ATA	2017-12-31	2018-01-02	1068	105.44s	31.56%	1.047GB
TLSE00FRA	2019-12-31	2020-01-02	1095	100.71s	32.04%	1.031GB
MAS100ESP	2023-12-31	2024-01-02	1095	134.30s	32.96%	1.697GB
Total Average	-	-	1086	113.48s	32.19%	1.258GB

Table A.8: Query Results for two days for TileDB station, and year.

Station	**Start Epoch**	**End Epoch**	**Number of Fragments**	**Time (s)**	**Avg CPU Usage** (%)	**Total Memory (GB)**
CAS100ATA	2017-12-31	2018-01-02	93	100.91s	32.52%	1.039GB
TLSE00FRA	2019-12-31	2020-01-02	93	102.43s	32.48%	1.036GB
MAS100ESP	2023-12-31	2024-01-02	93	131.86s	33.60%	1.695GB
Total Average	-	-	93.00	111.73s	32.87%	1.257GB

Table A.9: Query Results for two days for TileDB station, year, and month.